Adatbázisok elmélete

Adatbáziskezelő rendszerek

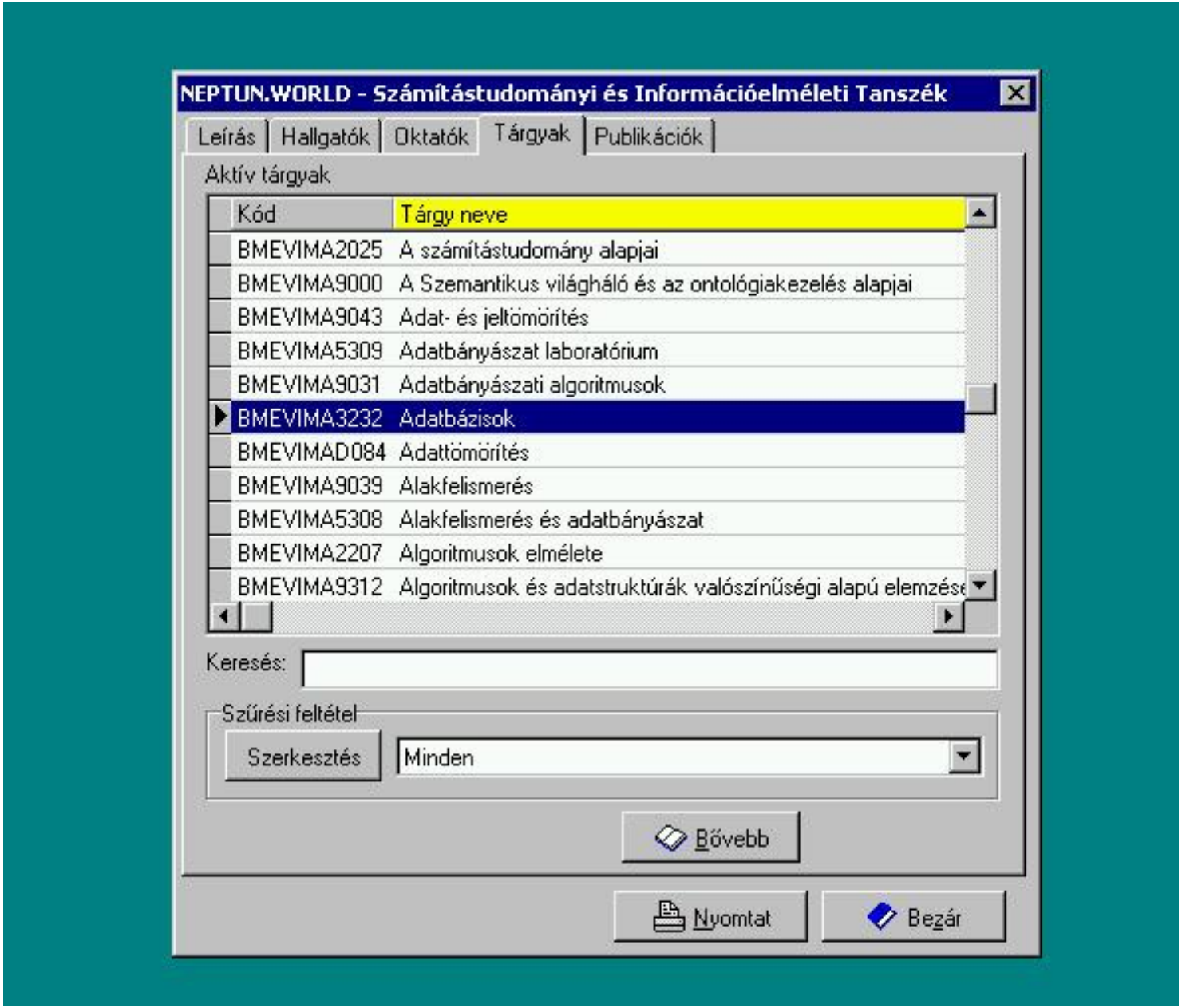
Katona Gyula Y.

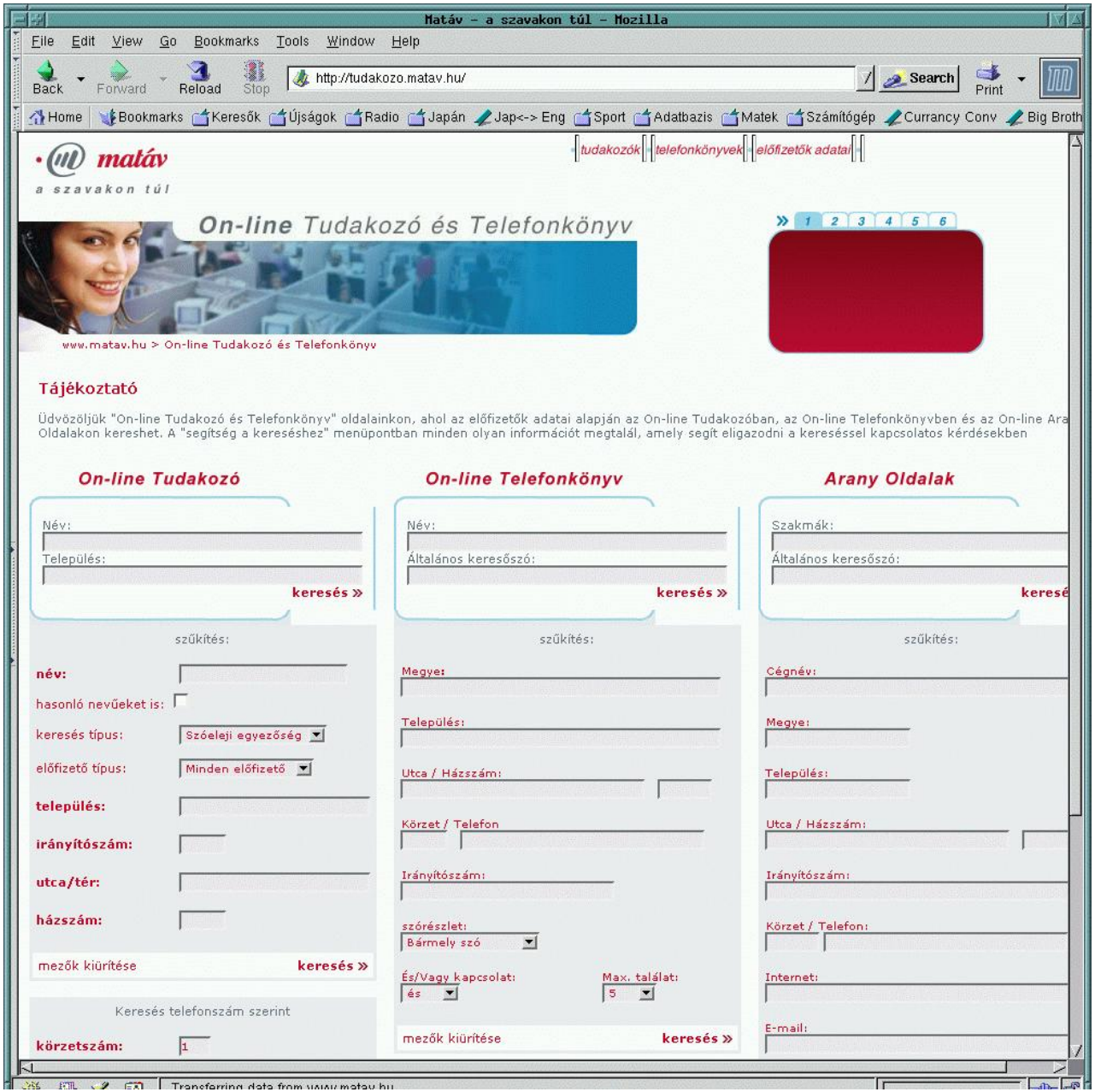
Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

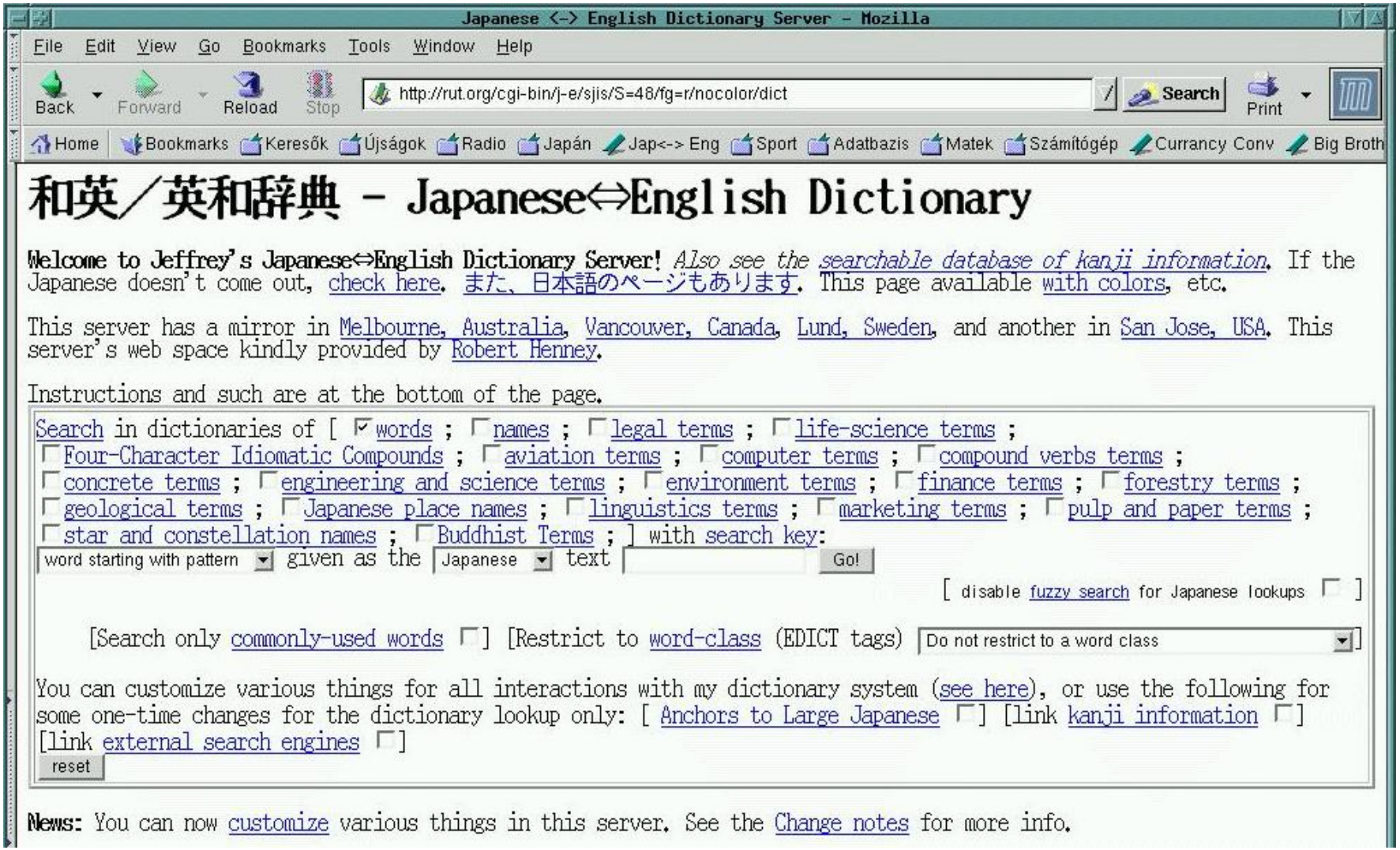
1. előadás

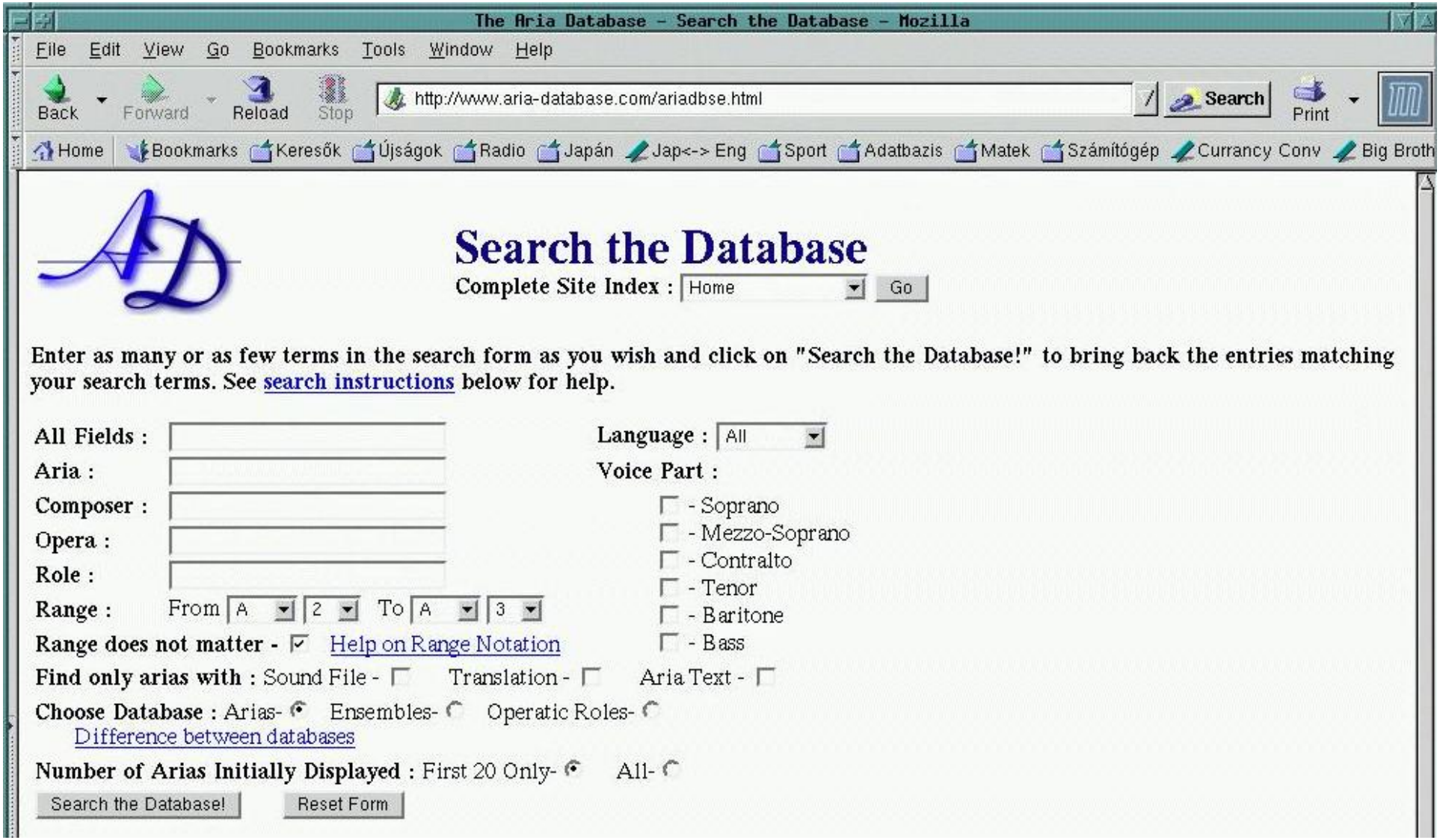
Adatbázis

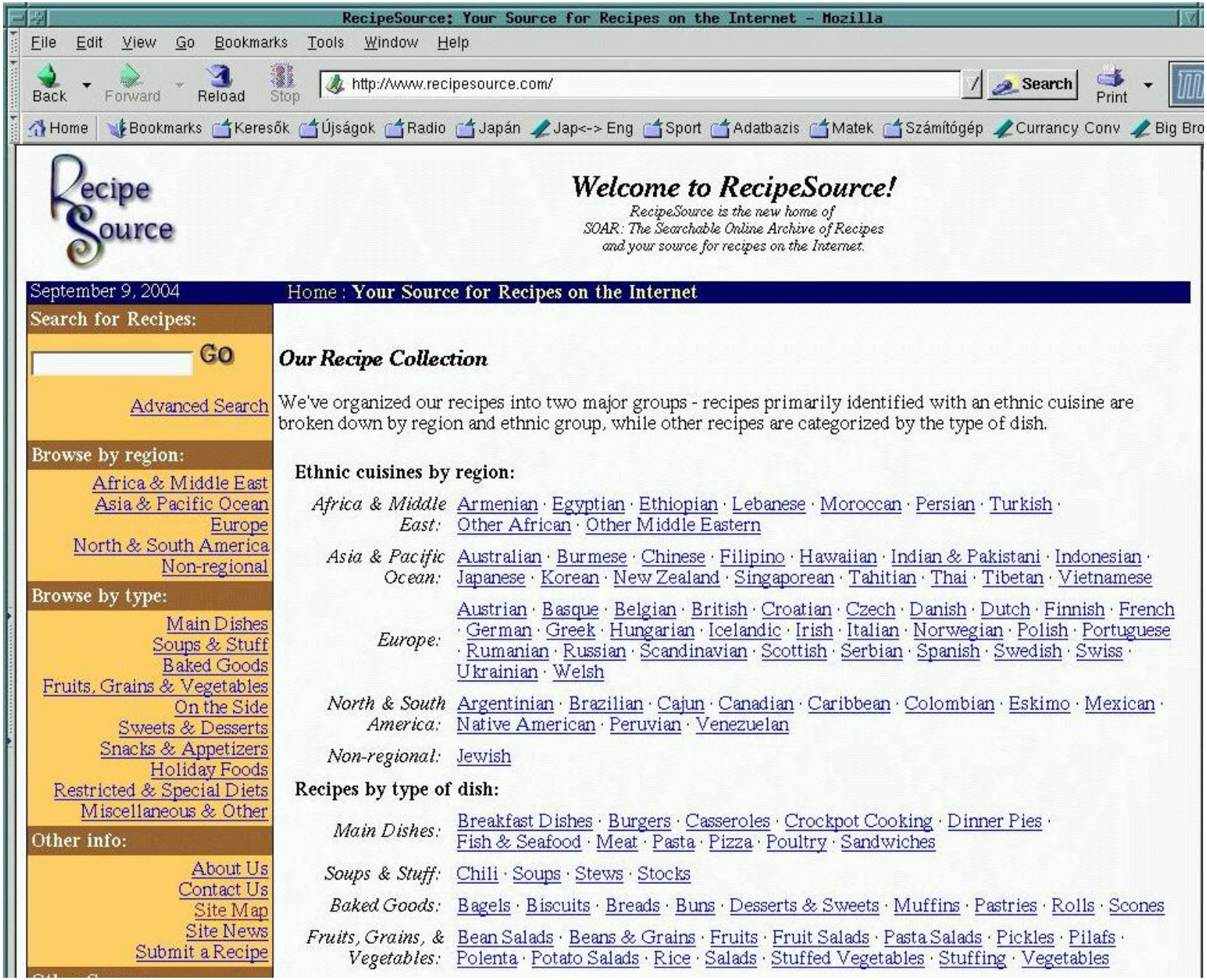
telefonkönyv, könyvtár, notesz, internet


NEPTUN.WORLD - Számítástudományi és Információelméleti Tanszék
Leírás
Hallgatók
Oktatók
Tárgyak
Publikációk
Aktív tárgyak
Kód
Tárgy neve
BMEVIMA2025 A számítástudomány alapjai
BMEVIMA9000 A Szemantikus világháló és az ontológiakezelés alapjai
BMEVIMA9043 Adat- és jeltömörítés
BMEVIMA5309 Adatbányászat laboratórium
BMEVIMA9031 Adatbányászati algoritmusok
BMEVIMA3232 Adatbázisok
BMEVIMAD084 Adattömörítés
BMEVIMA9039 Alakfelismerés
BMEVIMA5308 Alakfelismerés és adatbányászat
BMEVIMA2207 Algoritmusok elmélete
BMEVIMA9312 Algoritmusok és adatstruktúrák valószínűségi alapú elemzése
Keresés:
Szűrési feltétel
Szerkesztés
Minden
Bővebb
Nyomtat
Bezár


Matáv - a szavakon túl - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://tudakozo.matav.hu/
Search
Print
Home Bookmarks Keresők Újságok Radio Japán Jap<-> Eng Sport Adatbazis Matek Számítógép Currancy Conv Big Broth
matáv
a szavakon túl
tudakozók telefonkönyvek előfizetők adatai
On-line Tudakozó és Telefonkönyv
www.matav.hu > On-line Tudakozó és Telefonkönyv
Tájékoztató
Üdvözöljük "On-line Tudakozó és Telefonkönyv" oldalainkon, ahol az előfizetők adatai alapján az On-line Tudakozóban, az On-line Telefonkönyvben és az On-line Ara
Oldalakon kereshet. A "segítség a kereséshez" menüpontban minden olyan információt megtalál, amely segít eligazodni a kereséssel kapcsolatos kérdésekben
On-line Tudakozó
Név:
Település:
keresés »
szűkítés:
név:
hasonló nevűeket is:
keresés típus:
Szóeleji egyezőség
előfizető típus:
Minden előfizető
település:
irányítószám:
utca/tér:
házszám:
mezők kiürítése
keresés »
Keresés telefonszám szerint
körzetszám:
On-line Telefonkönyv
Név:
Általános keresőszó:
keresés »
szűkítés:
Megye:
Település:
Utca / Házszám:
Körzet / Telefon
Irányítószám:
szórészlet:
Bármely szó
És/Vagy kapcsolat:
és
Max. találat:
mezők kiürítése
keresés »
Arany Oldalak
Szakmák:
Általános keresőszó:
szűkítés:
Cégnév:
Megye:
Település:
Utca / Házszám:
Irányítószám:
Körzet / Telefon:
Internet:
E-mail:


Japanese <-> English Dictionary Server - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://rut.org/cgi-bin/j-e/sjis/S=48/fg=r/nocolor/dict
Search Print
Home Bookmarks Keresők Újságok Radio Japán Jap<-> Eng Sport Adatbazis Matek Számítógép Currancy Conv Big Broth
和英／英和辞典 - Japanese⇔English Dictionary
Welcome to Jeffrey's Japanese⇔English Dictionary Server! Also see the searchable database of kanji information. If the Japanese doesn't come out, check here. また、日本語のページもあります. This page available with colors, etc.
This server has a mirror in Melbourne, Australia, Vancouver, Canada, Lund, Sweden, and another in San Jose, USA. This server's web space kindly provided by Robert Henney.
Instructions and such are at the bottom of the page.
Search in dictionaries of [words ; names ; legal terms ; life-science terms ;
Four-Character Idiomatic Compounds ; aviation terms ; computer terms ; compound verbs terms ;
concrete terms ; engineering and science terms ; environment terms ; finance terms ; forestry terms ;
geological terms ; Japanese place names ; linguistics terms ; marketing terms ; pulp and paper terms ;
star and constellation names ; Buddhist Terms ;] with search key:
word starting with pattern given as the Japanese text Go!
[disable fuzzy search for Japanese lookups]
[Search only commonly-used words] [Restrict to word-class (EDICT tags) Do not restrict to a word class]
You can customize various things for all interactions with my dictionary system (see here), or use the following for some one-time changes for the dictionary lookup only: [Anchors to Large Japanese] [link kanji information]
[link external search engines]
reset
News: You can now customize various things in this server. See the Change notes for more info.


The Aria Database - Search the Database - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.aria-database.com/ariadbse.html
Search
Print
Home Bookmarks Keresők Újságok Radio Japán Jap<-> Eng Sport Adatbazis Matek Számítógép Currancy Conv Big Broth
Search the Database
Complete Site Index : Home Go
Enter as many or as few terms in the search form as you wish and click on "Search the Database!" to bring back the entries matching your search terms. See search instructions below for help.
All Fields :
Aria :
Composer :
Opera :
Role :
Range : From A 2 To A 3
Range does not matter - Help on Range Notation
Language : All
Voice Part :
- Soprano
- Mezzo-Soprano
- Contralto
- Tenor
- Baritone
- Bass
Find only arias with : Sound File - Translation - Aria Text -
Choose Database : Arias- Ensembles- Operatic Roles-
Difference between databases
Number of Arias Initially Displayed : First 20 Only- All-
Search the Database! Reset Form


RecipeSource: Your Source for Recipes on the Internet - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.recipesource.com/
Search
Print
Home Bookmarks Keresők Újságok Radio Japán Jap<-> Eng Sport Adatbazis Matek Számítógép Currancy Conv Big Bro
Recipe Source
Welcome to RecipeSource!
RecipeSource is the new home of
SOAR: The Searchable Online Archive of Recipes
and your source for recipes on the Internet.
September 9, 2004
Home : Your Source for Recipes on the Internet
Search for Recipes:
GO
Advanced Search
Browse by region:
Africa & Middle East
Asia & Pacific Ocean
Europe
North & South America
Non-regional
Browse by type:
Main Dishes
Soups & Stuff
Baked Goods
Fruits, Grains & Vegetables
On the Side
Sweets & Desserts
Snacks & Appetizers
Holiday Foods
Restricted & Special Diets
Miscellaneous & Other
Other info:
About Us
Contact Us
Site Map
Site News
Submit a Recipe
Our Recipe Collection
We've organized our recipes into two major groups - recipes primarily identified with an ethnic cuisine are broken down by region and ethnic group, while other recipes are categorized by the type of dish.
Ethnic cuisines by region:
Africa & Middle East: Armenian · Egyptian · Ethiopian · Lebanese · Moroccan · Persian · Turkish · Other African · Other Middle Eastern
Asia & Pacific Ocean: Australian · Burmese · Chinese · Filipino · Hawaiian · Indian & Pakistani · Indonesian · Japanese · Korean · New Zealand · Singaporean · Tahitian · Thai · Tibetan · Vietnamese
Europe: Austrian · Basque · Belgian · British · Croatian · Czech · Danish · Dutch · Finnish · French · German · Greek · Hungarian · Icelandic · Irish · Italian · Norwegian · Polish · Portuguese · Rumanian · Russian · Scandinavian · Scottish · Serbian · Spanish · Swedish · Swiss · Ukrainian · Welsh
North & South America: Argentinian · Brazilian · Cajun · Canadian · Caribbean · Colombian · Eskimo · Mexican · Native American · Peruvian · Venezuelan
Non-regional: Jewish
Recipes by type of dish:
Main Dishes: Breakfast Dishes · Burgers · Casseroles · Crockpot Cooking · Dinner Pies · Fish & Seafood · Meat · Pasta · Pizza · Poultry · Sandwiches
Soups & Stuff: Chili · Soups · Stews · Stocks
Baked Goods: Bagels · Biscuits · Breads · Buns · Desserts & Sweets · Muffins · Pastries · Rolls · Scones
Fruits, Grains, & Vegetables: Bean Salads · Beans & Grains · Fruits · Fruit Salads · Pasta Salads · Pickles · Pilafs · Polenta · Potato Salads · Rice · Salads · Stuffed Vegetables · Stuffing · Vegetables

Adatbáziskezelő rendszerek (DBMS) jellemzői

- Komplex hardver–szoftver rendszer adatok kezelésére (tárolás, módosítás, lekérdezés)

Adatbáziskezelő rendszerek (DBMS) jellemzői

- Komplex hardver–szoftver rendszer adatok kezelésére (tárolás, módosítás, lekérdezés)
- Nagy adatmennyiség

Adatbáziskezelő rendszerek (DBMS) jellemzői

- Komplex hardver–szoftver rendszer adatok kezelésére (tárolás, módosítás, lekérdezés)
- Nagy adatmennyiség
- Gazdag logikai struktúra (adatmodell) $\Longrightarrow$ magas szintű lekérdezés, módosítás (például egy légitársaságos adatbázisnál kategóriák lehetnek: utas, járat, pilóta)

Adatbáziskezelő rendszerek (DBMS) jellemzői

- Komplex hardver–szoftver rendszer adatok kezelésére (tárolás, módosítás, lekérdezés)
- Nagy adatmennyiség
- Gazdag logikai struktúra (adatmodell) $\Longrightarrow$ magas szintű lekérdezés, módosítás (például egy légitársaságos adatbázisnál kategóriák lehetnek: utas, járat, pilóta)
- hosszú életciklus (jól kell tűrnie a hardver módosításait, a fizikai tárolás változásait)

Elvárások a DBMS-mel szemben

- Új séma létrehozása (fogalmak, metaadatok megadása) pl.: legyen egy *utas* kategória, ennek attribútumai legyenek: név, lakcím, vegetáriánus-e,. . .
  Ehhez adott egy rendszertől függő DDL (Data Definition Language)

Elvárások a DBMS-mel szemben

- Új séma létrehozása (fogalmak, metaadatok megadása) pl.: legyen egy *utas* kategória, ennek attribútumai legyenek: név, lakcím, vegetáriánus-e,. . .
  Ehhez adott egy rendszertől függő DDL (Data Definition Language)
- Adatok beillesztése, módosítása (az adatbázis fogalmi keretének feltöltése) pl.: új utas felvétele, elromlott gép törlése . . .
  Ehhez adott egy rendszertől függő DML (Data Manipulation Language)

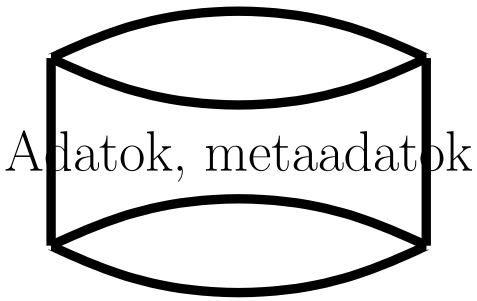
Elvárások a DBMS-mel szemben

- Új séma létrehozása (fogalmak, metaadatok megadása) pl.: legyen egy *utas* kategória, ennek attribútumai legyenek: név, lakcím, vegetáriánus-e,. . .
  Ehhez adott egy rendszertől függő DDL (Data Definition Language)
- Adatok beillesztése, módosítása (az adatbázis fogalmi keretének feltöltése) pl.: új utas felvétele, elromlott gép törlése . . .
  Ehhez adott egy rendszertől függő DML (Data Manipulation Language)
- Lekérdezés (infót kiszedni az adatbázisból)
  Ehhez Query Language (pl.: SQL)
  Cél: gyakori kérdéseket könnyű legyen kérdezni (ehhez jól végiggondolt fogalmi keret kell, hatékony tárolás $\Longrightarrow$ fontos a jó tervezés)
- Nagy mennyiségű adat tárolása biztonságosan (jogosultságok, illetéktelen hozzáférés megakadályozása illetve rendszerhibák elleni védelem)

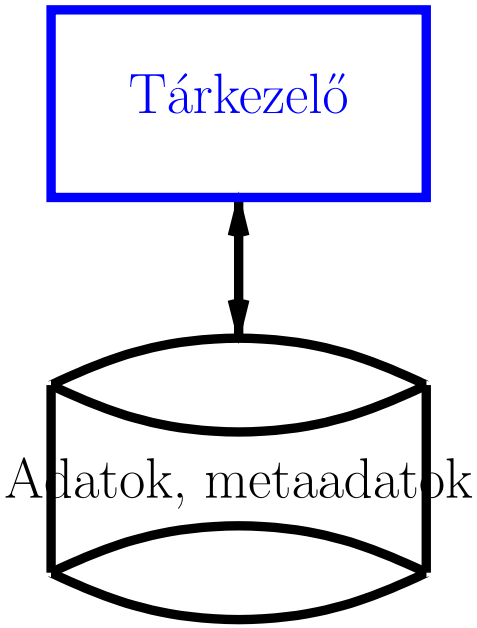
Elvárások a DBMS-mel szemben

- Új séma létrehozása (fogalmak, metaadatok megadása) pl.: legyen egy *utas* kategória, ennek attribútumai legyenek: név, lakcím, vegetáriánus-e,. . .
  Ehhez adott egy rendszertől függő DDL (Data Definition Language)
- Adatok beillesztése, módosítása (az adatbázis fogalmi keretének feltöltése) pl.: új utas felvétele, elromlott gép törlése . . .
  Ehhez adott egy rendszertől függő DML (Data Manipulation Language)
- Lekérdezés (infót kiszedni az adatbázisból)
  Ehhez Query Language (pl.: SQL)
  Cél: gyakori kérdéseket könnyű legyen kérdezni (ehhez jól végiggondolt fogalmi keret kell, hatékony tárolás $\Longrightarrow$ fontos a jó tervezés)
- Nagy mennyiségű adat tárolása biztonságosan (jogosultságok, illetéktelen hozzáférés megakadályozása illetve rendszerhibák elleni védelem)
- Többfelhasználós működés támogatása (egyidejű hozzáférés, de ne legyen hibás, inkonzisztens állapot emiatt)

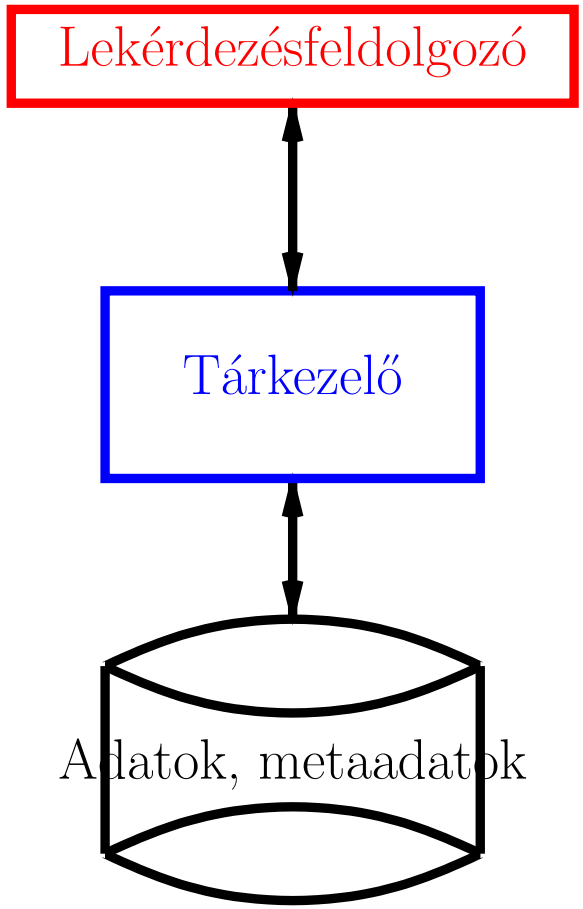
A DBMS felépítése, részei



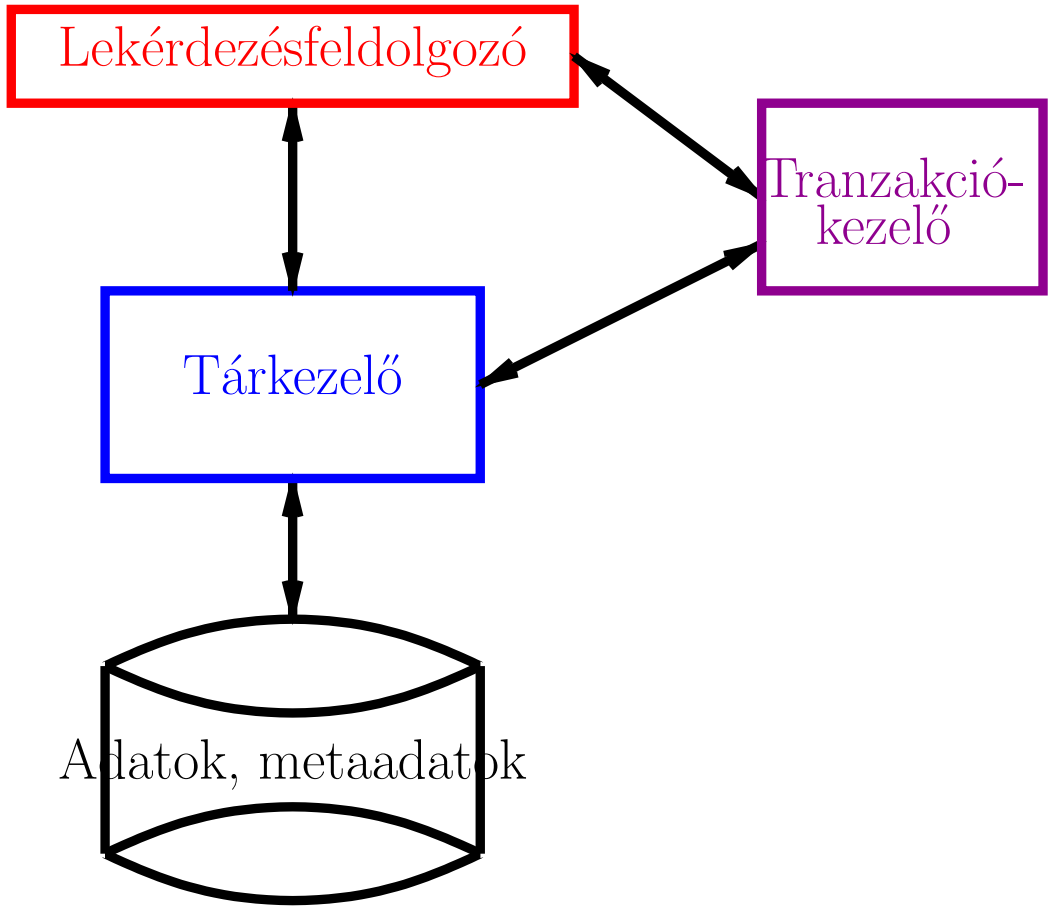
A DBMS felépítése, részei



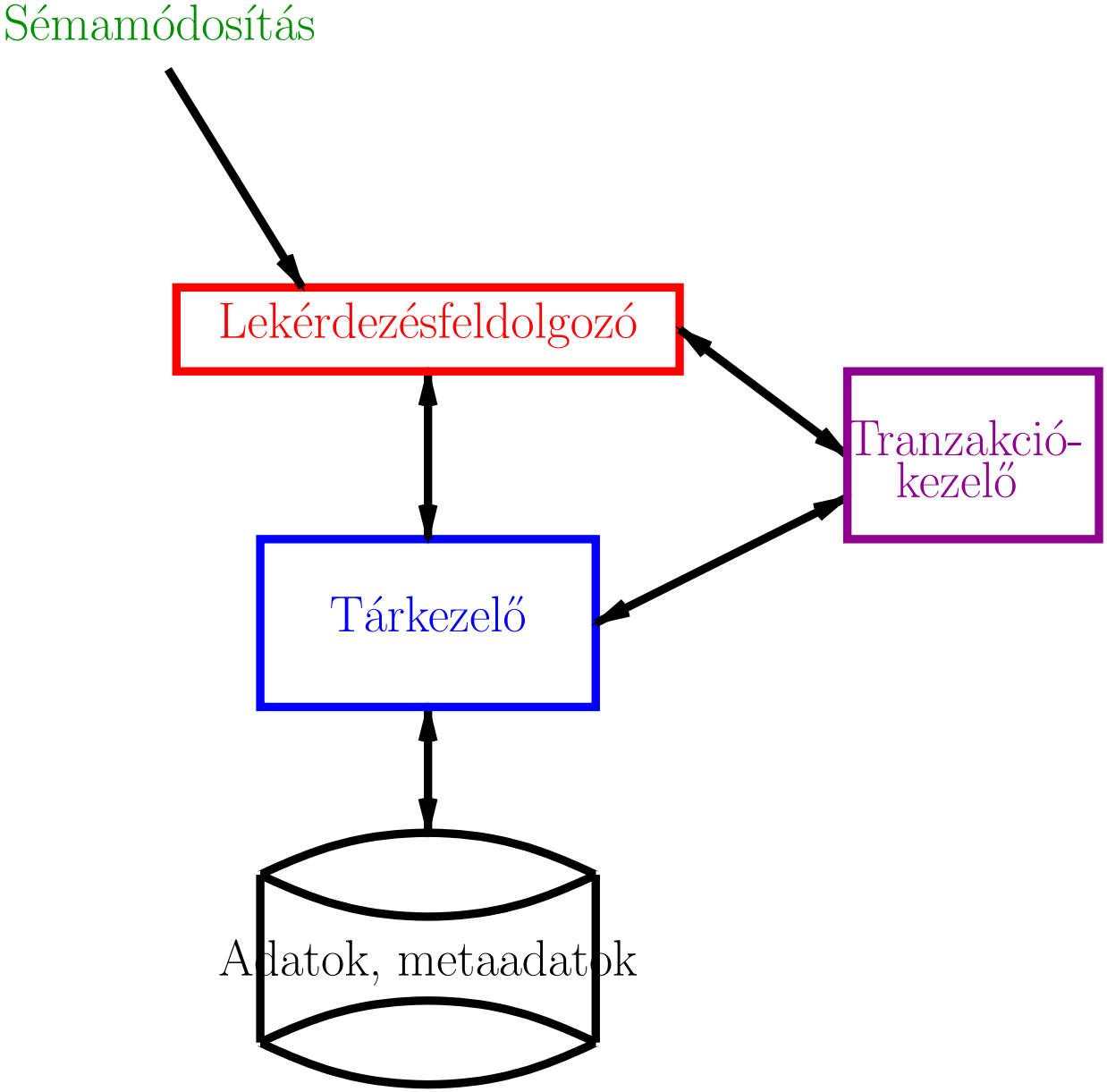
A DBMS felépítése, részei



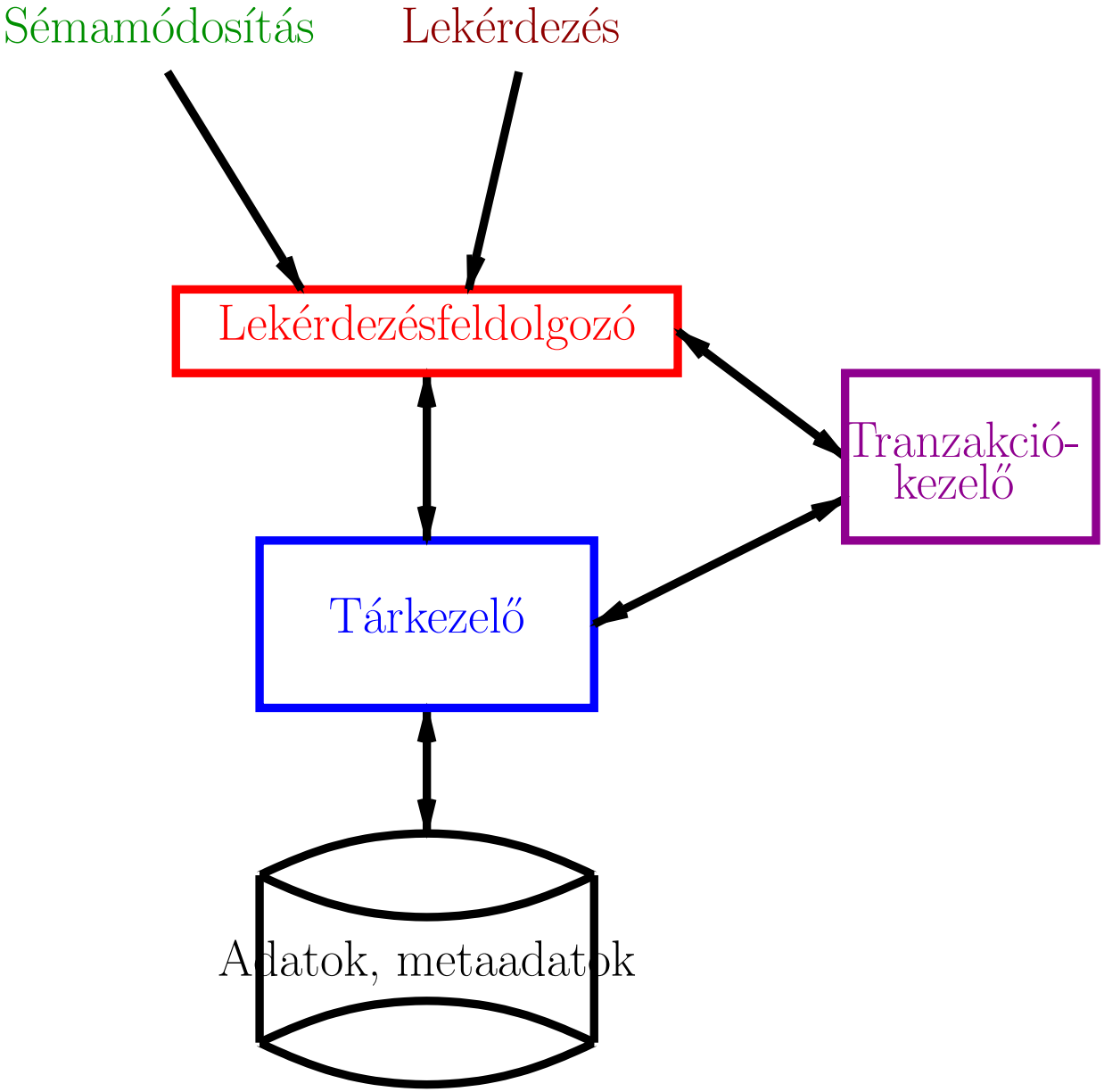
A DBMS felépítése, részei



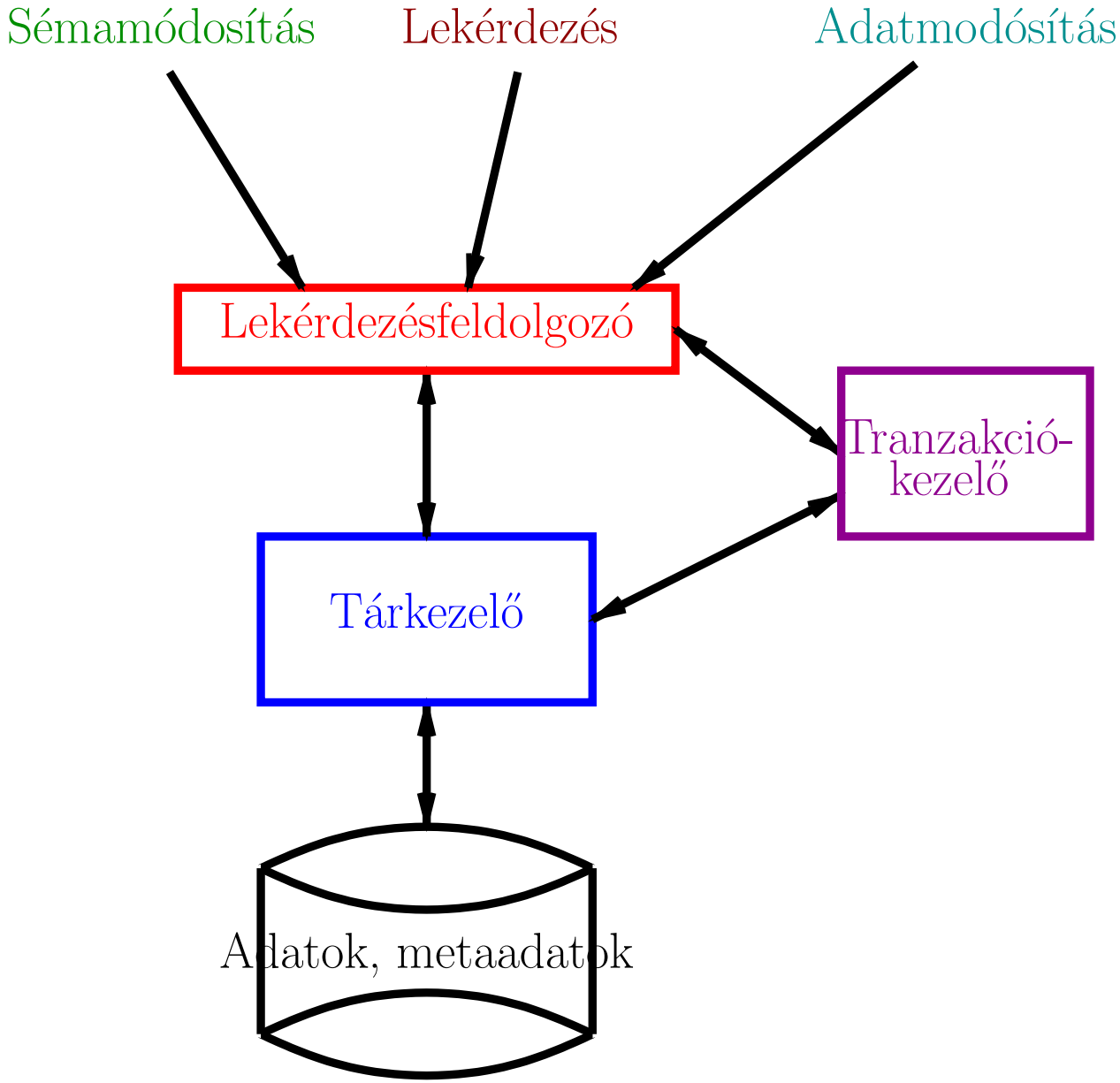
A DBMS felépítése, részei



A DBMS felépítése, részei



A DBMS felépítése, részei



Adatok, metaadatok

Fizikailag valahol tárolódnak az adatok (milyen nevű utas, melyik gépre foglalt helyet) és a metaadatok (mik a relációk nevei és attribútumai és ezek típusai, illetve pl. milyen indexek vannak a kereséshez)

Adatok, metaadatok

Fizikailag valahol tárolódnak az adatok (milyen nevű utas, melyik gépre foglalt helyet) és a metaadatok (mik a relációk nevei és attribútumai és ezek típusai, illetve pl. milyen indexek vannak a kereséshez)

Tárkezelő

A kért információ beolvasása, módosítása. Kb. mint az OR-ek file kezelője, de itt néha (többfelhasználós működésnél pl.) mi mondjuk meg, hova történjen az írás (háttártárra, pufferbe), nem kezelheti teljesen szabadon a puffert.

Adatok, metaadatok

Fizikailag valahol tárolódnak az adatok (milyen nevű utas, melyik gépre foglalt helyet) és a metaadatok (mik a relációk nevei és attribútumai és ezek típusai, illetve pl. milyen indexek vannak a kereséshez)

Tárkezelő

A kért információ beolvasása, módosítása. Kb. mint az OR-ek file kezelője, de itt néha (többfelhasználós működésnél pl.) mi mondjuk meg, hova történjen az írás (háttártárra, pufferbe), nem kezelheti teljesen szabadon a puffert.
Részei:

- File kezelő: nyilvántartja a DB állományát; fizikai I/O-t végez, ha a pufferkezelő kéri; indexstruktúrába rendezni az adatokat (pl. B-fa) (I/O mindig lemezblokkonként, hatékonyság mértéke az ilyen műveletek száma)

Adatok, metaadatok

Fizikailag valahol tárolódnak az adatok (milyen nevű utas, melyik gépre foglalt helyet) és a metaadatok (mik a relációk nevei és attribútumai és ezek típusai, illetve pl. milyen indexek vannak a kereséshez)

Tárkezelő

A kért információ beolvasása, módosítása. Kb. mint az OR-ek file kezelője, de itt néha (többfelhasználós működésnél pl.) mi mondjuk meg, hova történjen az írás (háttártárra, pufferbe), nem kezelheti teljesen szabadon a puffert.
Részei:

- File kezelő: nyilvántartja a DB állományát; fizikai I/O-t végez, ha a pufferkezelő kéri; indexstruktúrába rendezni az adatokat (pl. B-fa) (I/O mindig lemezblokkonként, hatékonyság mértéke az ilyen műveletek száma)
- Pufferkezelő: kezeli a memóriát, tárolja a filekezelő által beolvasott blokkokat

Lekérdezésfeldolgozó

Alapfeladata: sémadefiníciós, adatmódosítás és lekérdezős kérések fogadása, kezelése

Lekérdezésfeldolgozó

Alapfeladata: sémadefiníciós, adatmódosítás és lekérdezős kérések fogadása, kezelése

- Sémaműveletek: a DB logikai struktúrájának kialakítása, módosítása
  eredménye: maga az adatbázisséma, plusz kiegészítő metaadatok (pl. mit hogyan tároljunk, indexek)
  Nagyon fontos, meghatározza a további működést

Lekérdezésfeldolgozó

Alapfeladata: sémadefiníciós, adatmódosítás és lekérdezős kérések fogadása, kezelése

- Sémaműveletek: a DB logikai struktúrájának kialakítása, módosítása
  eredménye: maga az adatbázisséma, plusz kiegészítő metaadatok (pl. mit hogyan tároljunk, indexek)
  Nagyon fontos, meghatározza a további működést
- Lekérdezések: keresőkérdések a DB-re vonatkozóan
  két lehetőség erre: vagy egy külön lekérdezőfelületen át vagy alkalmazói programból (a második esetben a host language-nek van utasításkészlete a DB-hez fordulásra)

Lekérdezésfeldolgozó

Alapfeladata: sémadefiníciós, adatmódosítás és lekérdezős kérések fogadása, kezelése

- Sémaműveletek: a DB logikai struktúrájának kialakítása, módosítása
  eredménye: maga az adatbázisséma, plusz kiegészítő metaadatok (pl. mit hogyan tároljunk, indexek)
  Nagyon fontos, meghatározza a további működést
- Lekérdezések: keresőkérdések a DB-re vonatkozóan
  két lehetőség erre: vagy egy külön lekérdezőfelületen át vagy alkalmazói programból (a második esetben a host language-nek van utasításkészlete a DB-hez fordulásra)
- Adatmódosítás: a DB tartamának módosítása, beszúrás, törlés
  Itt is lehet külön felület vagy program

Lekérdezésfeldolgozó tennivalói

Hogyan kezeli ezeket a kéréseket?

Lekérdezésfeldolgozó tennivalói

Hogyan kezeli ezeket a kéréseket?

1. Végrehajtási terv készítése

A magas szintű kérdéseket átalakítjuk elemi utasítások sorozatává.
A (legtöbbször) deklaratív kérdésből (ahol nem mondjuk meg, hogy milyen úton akarom megkapni az eredményt, csak azt, hogy mit akarok, pl. SQL) procedurális kérdést csinálunk (ahol már egy konkrét végrehajtási terv látszik).

Lekérdezésfeldolgozó tennivalói

Hogyan kezeli ezeket a kéréseket?

1. Végrehajtási terv készítése

A magas szintű kérdéseket átalakítjuk elemi utasítások sorozatává.
A (legtöbbször) deklaratív kérdésből (ahol nem mondjuk meg, hogy milyen úton akarom megkapni az eredményt, csak azt, hogy mit akarok, pl. SQL) procedurális kérdést csinálunk (ahol már egy konkrét végrehajtási terv látszik).

Pl: Ha egy banki séma két relációja
ügyfél(név, cím, szemszám) és
számla(szemszám, számlaszám, egyenleg)
és mi Bill Gates számlájának egyelegét szeretnénk megkapni, akkor erre egy SQL kérdés:

SENECÁT egyenleg FROM ügyfél, számla
WHERE ügyfél.szemszám=számla.szemszám AND ügyfél.név=”Bill Gates”

Pl: Ha egy banki séma két relációja
ügyfél(név, cím, szemszám) és
számla(szemszám, számlaszám, egyenleg)
és mi Bill Gates számlájának egyelegét szeretnénk megkapni, akkor erre egy SQL kérdés:

SENECÁT egyenleg FROM ügyfél, számla
WHERE ügyfél.szemszám=számla.szemszám AND ügyfél.név=”Bill Gates”

Ez csak azt mondja meg, hogy mely relációkból, mit akarok megkapni, de azt nem, hogy hogyan kell ezt megszerezni. Erre egy (elnagyolt) végrehajtási terv lehet pl. az, hogy ha van index a névre az *ügyfél*-ben, akkor az alapján keressük meg B.G. személyi számát, aztán ez alapján a másik relációban keressük meg a megfelelő sort és olvassuk ki az egyenleget.

2. „Optimalizálás”

Több lehetséges végrehajtási terv közül kiválasztani egy „legjobbat”. Nem valódi optimalizálás, nem a legjobbat keressük, csak egy elég jót.

2. „Optimalizálás”

Több lehetséges végrehajtási terv közül kiválasztani egy „legjobbat”. Nem valódi optimalizálás, nem a legjobbat keressük, csak egy elég jót.

Tanulság:

1. A kapott kérdés nem ad támpontot, hogy hogyan kell megkapni az eredményt, de nem is kötelez semmire.

2. „Optimalizálás”

Több lehetséges végrehajtási terv közül kiválasztani egy „legjobbat”. Nem valódi optimalizálás, nem a legjobbat keressük, csak egy elég jót.

Tanulság:

1. A kapott kérdés nem ad támpontot, hogy hogyan kell megkapni az eredményt, de nem is kötelez semmire.
2. Érdemes szöszölni egy jobb végrehajtási terv keresésével, mert nagyok lehetnek a különbségek.

Tranzakciókezelő

Két nagyobb problémakör megoldására jó:

- Több felhasználó egyszerre használja a DB-t, egyidejű hozzáférések kezelése

Tranzakciókezelő

Két nagyobb problémakör megoldására jó:

- Több felhasználó egyszerre használja a DB-t, egyidejű hozzáférések kezelése
- Rendszerhibák, ABORT-ok hatásainak kivédése: ezek bekövetkeztekor sem veszhetnek el adatok, nem maradhat a DB inkonzisztens állapotban

Tranzakciókezelő

Két nagyobb problémakör megoldására jó:

- Több felhasználó egyszerre használja a DB-t, egyidejű hozzáférések kezelése
- Rendszerhibák, ABORT-ok hatásainak kivédése: ezek bekövetkeztekor sem veszhetnek el adatok, nem maradhat a DB inkonzisztens állapotban

Ezek megoldására: alapfogalom a
tranzakció: egy felhasználóhoz tartozó, összetartozó utasítások olyan sorozata, melyek vagy mind végrehajtódnak vagy semelyik sem (ez a tulajdonság az atomiság)

Tranzakciókezelő

Két nagyobb problémakör megoldására jó:

- Több felhasználó egyszerre használja a DB-t, egyidejű hozzáférések kezelése
- Rendszerhibák, ABORT-ok hatásainak kivédése: ezek bekövetkeztekor sem veszhetnek el adatok, nem maradhat a DB inkonzisztens állapotban

Ezek megoldására: alapfogalom a
tranzakció: egy felhasználóhoz tartozó, összetartozó utasítások olyan sorozata, melyek vagy mind végrehajtódnak vagy semelyik sem (ez a tulajdonság az atomiság)
Pl. banki átutalásnál nem lehet, hogy csak a pénz levonása történik meg az egyik számlán, de nem íródik jóvá a másikon

Elvárások a tranzakciókezelésben

- A (atomicity, atomiság): egy tranzakció vagy teljesen végrehajtódik vagy semmi se hajtódik végre belőle

Elvárások a tranzakciókezelésben

- A (atomicity, atomiság): egy tranzakció vagy teljesen végrehajtódik vagy semmi se hajtódik végre belőle
- C (consistency, konzisztencia): vannak a rendszernek helyes, konzisztens állapotai, ezek között mozog. Inkonzisztens állapot csak ideiglenesen lehet.

Elvárások a tranzakciókezelésben

- A (atomicity, atomiság): egy tranzakció vagy teljesen végrehajtódik vagy semmi se hajtódik végre belőle
- C (consistency, konzisztencia): vannak a rendszernek helyes, konzisztens állapotai, ezek között mozog. Inkonzisztens állapot csak ideiglenesen lehet.
- I (isolation, elkülönítés): több tranzakció egyidejű lefutása után úgy nézzen ki a rendszer, mintha a tranzakciók egymás után, elkülönülten futottak volna le.

Elvárások a tranzakciókezelésben

- A (atomicity, atomiság): egy tranzakció vagy teljesen végrehajtódik vagy semmi se hajtódik végre belőle
- C (consistency, konzisztencia): vannak a rendszernek helyes, konzisztens állapotai, ezek között mozog. Inkonzisztens állapot csak ideiglenesen lehet.
- I (isolation, elkülönítés): több tranzakció egyidejű lefutása után úgy nézzen ki a rendszer, mintha a tranzakciók egymás után, elkülönülten futottak volna le. Tehát pl. az alábbi párhuzamos lefutás:

hatása legyen ugyanaz, mint vagy a tr1 tr2 sorrend, vagy a tr2 tr1 sorrend hatása.

Elvárások a tranzakciókezelésben

- A (atomicity, atomiság): egy tranzakció vagy teljesen végrehajtódik vagy semmi se hajtódik végre belőle
- C (consistency, konzisztencia): vannak a rendszernek helyes, konzisztens állapotai, ezek között mozog. Inkonzisztens állapot csak ideiglenesen lehet.
- I (isolation, elkülönítés): több tranzakció egyidejű lefutása után úgy nézzen ki a rendszer, mintha a tranzakciók egymás után, elkülönülten futottak volna le. Tehát pl. az alábbi párhuzamos lefutás:

  hatása legyen ugyanaz, mint vagy a tr1 tr2 sorrend, vagy a tr2 tr1 sorrend hatása.
- D (durability, tartósság): ha egy tranzakció sikeresen befejeződött, akkor annak eredménye nem veszhet el később.

Eszközök a tranzakciókezelésben- ízelítő

- **Zárolás**: ha egy tranzakció csinálni akar valamit egy adattal, akkor zárat rak rá, ekkor más nem, vagy csak korlátozottan fér hozzá (lehet több féle zárat is használni). Zárolni lehet sort, blokkot, egész táblát is: minél nagyobbat zárolunk, annál könnyebb az adminisztráció, de annál többet kell a tranzakcióknak várniuk egymásra.

Eszközök a tranzakciókezelésben- ízelítő

- **Zárolás**: ha egy tranzakció csinálni akar valamit egy adattal, akkor zárat rak rá, ekkor más nem, vagy csak korlátozottan fér hozzá (lehet több féle zárat is használni). Zárolni lehet sort, blokkot, egész táblát is: minél nagyobbat zárolunk, annál könnyebb az adminisztráció, de annál többet kell a tranzakcióknak várniuk egymásra.
- **Naplózás**: van egy biztos, hibáktól védett helyen tartott napló, ahol minden történést feljegyzünk. Hiba esetén ez megmarad, innen vissza lehet állítani egy helyes állapotot.

Eszközök a tranzakciókezelésben- ízelítő

- **Zárolás**: ha egy tranzakció csinálni akar valamit egy adattal, akkor zárat rak rá, ekkor más nem, vagy csak korlátozottan fér hozzá (lehet több féle zárat is használni). Zárolni lehet sort, blokkot, egész táblát is: minél nagyobbat zárolunk, annál könnyebb az adminisztráció, de annál többet kell a tranzakcióknak várniuk egymásra.
- **Naplózás**: van egy biztos, hibáktól védett helyen tartott napló, ahol minden történést feljegyzünk. Hiba esetén ez megmarad, innen vissza lehet állítani egy helyes állapotot.

Ez az eddig felsorolt sok alkotórész eloszlik a kliens és a szerver között. A szerver tartja a kapcsolatot az fizikai adatbázissal, a kliens pedig a felhasználóval, a többi funkció eloszlása nagyon változhat rendszertől függően.

Az adatbáziskezelő használati szintjei

- Felhasználó: egyszerűbb SQL kérdést fogalmaz meg, adatmódosítást csinál, alkalmazói programot futtat

Az adatbáziskezelő használati szintjei

- Felhasználó: egyszerűbb SQL kérdést fogalmaz meg, adatmódosítást csinál, alkalmazói programot futtat
- Adatbázis programozó: összetett kérdések írása, alkalmazói program írása. Jól ismeri a DB felépítését, a DBMS-ben használt technikákat.

Az adatbáziskezelő használati szintjei

- Felhasználó: egyszerűbb SQL kérdést fogalmaz meg, adatmódosítást csinál, alkalmazói programot futtat
- Adatbázis programozó: összetett kérdések írása, alkalmazói program írása. Jól ismeri a DB felépítését, a DBMS-ben használt technikákat.
- Adatbázis tervező: sémát hoz létre, fizikai szervezésbe beleszólhat.

Az adatbáziskezelő használati szintjei

- Felhasználó: egyszerűbb SQL kérdést fogalmaz meg, adatmódosítást csinál, alkalmazói programot futtat
- Adatbázis programozó: összetett kérdések írása, alkalmazói program írása. Jól ismeri a DB felépítését, a DBMS-ben használt technikákat.
- Adatbázis tervező: sémát hoz létre, fizikai szervezésbe beleszólhat.
- Adatbázisrendszer megvalósító: az adatbáziskezelőt magát tervezi és írja.

A félév anyaga

Főleg a tervezés és megvalósítás kérdései:

A félév anyaga

Főleg a tervezés és megvalósítás kérdései:

- **Tervezés**:
  - Adatmodellezés: adatbázis-tervező technikák (ODL, objektumos tervezés és E/K (egyed-kapcsolat) diagram, grafikus jelölésrendszer).
    Az adatbázis fogalmi keretének megadására jók, tervet lehet velük készíteni, amit aztán majd át kell alakítani az adatbáziskezelő által használt formális megadási módra.

A félév anyaga

Főleg a tervezés és megvalósítás kérdései:

- **Tervezés**:
  - Adatmodellezés: adatbázis-tervező technikák (ODL, objektumos tervezés és E/K (egyed-kapcsolat) diagram, grafikus jelölésrendszer).
    Az adatbázis fogalmi keretének megadására jók, tervet lehet velük készíteni, amit aztán majd át kell alakítani az adatbáziskezelő által használt formális megadási módra.
  - Relációs adatmodell: nagyon fontos, tipikusan ilyenek a DBMS-ek mostanában.

A félév anyaga

Főleg a tervezés és megvalósítás kérdései:

- **Tervezés**:
  - Adatmodellezés: adatbázis-tervező technikák (ODL, objektumos tervezés és E/K (egyed-kapcsolat) diagram, grafikus jelölésrendszer).
    Az adatbázis fogalmi keretének megadására jók, tervet lehet velük készíteni, amit aztán majd át kell alakítani az adatbáziskezelő által használt formális megadási módra.
  - Relációs adatmodell: nagyon fontos, tipikusan ilyenek a DBMS-ek mostanában.
- **Tervezés/megvalósítás**:
  - fizikai szervezés

A félév anyaga

Főleg a tervezés és megvalósítás kérdései:

- **Tervezés**:
  - Adatmodellezés: adatbázis-tervező technikák (ODL, objektumos tervezés és E/K (egyed-kapcsolat) diagram, grafikus jelölésrendszer).
    Az adatbázis fogalmi keretének megadására jók, tervet lehet velük készíteni, amit aztán majd át kell alakítani az adatbáziskezelő által használt formális megadási módra.
  - Relációs adatmodell: nagyon fontos, tipikusan ilyenek a DBMS-ek mostanában.
- **Tervezés/megvalósítás**:
  - fizikai szervezés
  - tranzakciókezelés

A félév anyaga

Főleg a tervezés és megvalósítás kérdései:

- **Tervezés**:
  - Adatmodellezés: adatbázis-tervező technikák (ODL, objektumos tervezés és E/K (egyed-kapcsolat) diagram, grafikus jelölésrendszer).
    Az adatbázis fogalmi keretének megadására jók, tervet lehet velük készíteni, amit aztán majd át kell alakítani az adatbáziskezelő által használt formális megadási módra.
  - Relációs adatmodell: nagyon fontos, tipikusan ilyenek a DBMS-ek mostanában.
- **Tervezés/megvalósítás**:
  - fizikai szervezés
  - tranzakciókezelés
  - lekérdezésfeldolgozás

A félév anyaga

Főleg a tervezés és megvalósítás kérdései:

- **Tervezés**:
  - Adatmodellezés: adatbázis-tervező technikák (ODL, objektumos tervezés és E/K (egyed-kapcsolat) diagram, grafikus jelölésrendszer).
    Az adatbázis fogalmi keretének megadására jók, tervet lehet velük készíteni, amit aztán majd át kell alakítani az adatbáziskezelő által használt formális megadási módra.
  - Relációs adatmodell: nagyon fontos, tipikusan ilyenek a DBMS-ek mostanában.
- **Tervezés/megvalósítás**:
  - fizikai szervezés
  - tranzakciókezelés
  - lekérdezésfeldolgozás
- **Programozás**: nem nagyon lesz, majd laboron a következő félévben, de azért: SQL, adatmódosítás, adatdefiniálás, triggerek, megszorítások kezeléséről lesz szó.

Adatbázisok elmélete

Adatbáziskezelő rendszerekfelépítése, adatmodellezés, ODL

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

2. előadás

Adatbáziskezelő rendszerek története

Ősei a file-kezelők; ezek nem teljesítik ugyan azokat az elvárásokat, amiket a DBMS-sel szemben támasztunk, de sok a hasonlóság: sok adat, hosszú élettartam. Viszont primitív a lekérdezés (csak a file-hierarchiában lehet mozogni), nincs sémadefiníció (csak könyvtárszerkezet), nincs védelem rendszerhibák esetére, többfelhasználós működés sincs támogatva.

Adatbáziskezelő rendszerek története

Ősei a file-kezelők; ezek nem teljesítik ugyan azokat az elvárásokat, amiket a DBMS-sel szemben támasztunk, de sok a hasonlóság: sok adat, hosszú élettartam. Viszont primitív a lekérdezés (csak a file-hierarchiában lehet mozogni), nincs sémadefiníció (csak könyvtárszerkezet), nincs védelem rendszerhibák esetére, többfelhasználós működés sincs támogatva.

Első rendszerek

Jellemzők: sok kis adat, gyakori, de kevés adatot érintő lekérdezések, módosítások.

Első rendszerek

Jellemzők: sok kis adat, gyakori, de kevés adatot érintő lekérdezések, módosítások.

- *Repülőgépes helyfoglalás*
  Adatelemek: indulás, érkezés, honnan indul, hova érkezik, ár, darabszám, utas neve. . .
  Lekérdezések: van-e még hely, mennyi az ára, mikor indul a gép
  Módosítások: új utas bevitele, helyfoglalás
  Párhuzamosság: egyszerre több jegyeladás és lekérdezés
  Védelem: helyfoglalás nem veszhet el

Első rendszerek

Jellemzők: sok kis adat, gyakori, de kevés adatot érintő lekérdezések, módosítások.

- *Repülőgépes helyfoglalás*
  Adatelemek: indulás, érkezés, honnan indul, hova érkezik, ár, darabszám, utas neve. . .
  Lekérdezések: van-e még hely, mennyi az ára, mikor indul a gép
  Módosítások: új utas bevitele, helyfoglalás
  Párhuzamosság: egyszerre több jegyeladás és lekérdezés
  Védelem: helyfoglalás nem veszhet el
- *Banki rendszerek*
  Adatelemek: ügyfelek adatai, számlák adatai, jogosultságok. . .
  Lekérdezések: egyenlegek
  Módosítások: pénzmozgások
  Párhuzamosság, biztonság fontos/megoldva valahogy.

Első rendszerek

Jellemzők: sok kis adat, gyakori, de kevés adatot érintő lekérdezések, módosítások.

- *Repülőgépes helyfoglalás*
  Adatelemek: indulás, érkezés, honnan indul, hova érkezik, ár, darabszám, utas neve. . .
  Lekérdezések: van-e még hely, mennyi az ára, mikor indul a gép
  Módosítások: új utas bevitele, helyfoglalás
  Párhuzamosság: egyszerre több jegyeladás és lekérdezés
  Védelem: helyfoglalás nem veszhet el
- *Banki rendszerek*
  Adatelemek: ügyfelek adatai, számlák adatai, jogosultságok. . .
  Lekérdezések: egyenlegek
  Módosítások: pénzmozgások
  Párhuzamosság, biztonság fontos/megoldva valahogy.
- *Vállalati rendszerek*
  Ügyfelek, eladások, szerződések adatai, kimutatások készítése, új szerződések bevitele.

Első rendszerek

Jellemzők: sok kis adat, gyakori, de kevés adatot érintő lekérdezések, módosítások.

- *Repülőgépes helyfoglalás*
  Adatelemek: indulás, érkezés, honnan indul, hova érkezik, ár, darabszám, utas neve. . .
  Lekérdezések: van-e még hely, mennyi az ára, mikor indul a gép
  Módosítások: új utas bevitele, helyfoglalás
  Párhuzamosság: egyszerre több jegyeladás és lekérdezés
  Védelem: helyfoglalás nem veszhet el
- *Banki rendszerek*
  Adatelemek: ügyfelek adatai, számlák adatai, jogosultságok. . .
  Lekérdezések: egyenlegek
  Módosítások: pénzmozgások
  Párhuzamosság, biztonság fontos/megoldva valahogy.
- *Vállalati rendszerek*
  Ügyfelek, eladások, szerződések adatai, kimutatások készítése, új szerződések bevitele.

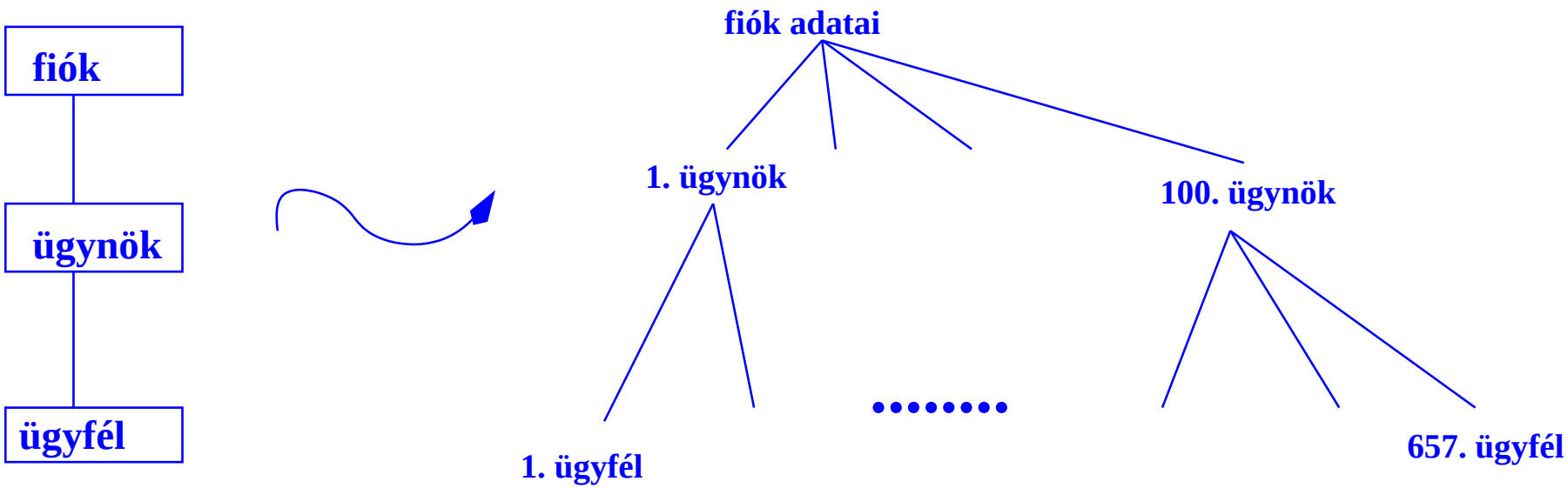
Korai modellek

Közös jellemzők: a fogalmi keret tükrözi a tárolást

Korai modellek

Közös jellemzők: a fogalmi keret tükrözi a tárolást

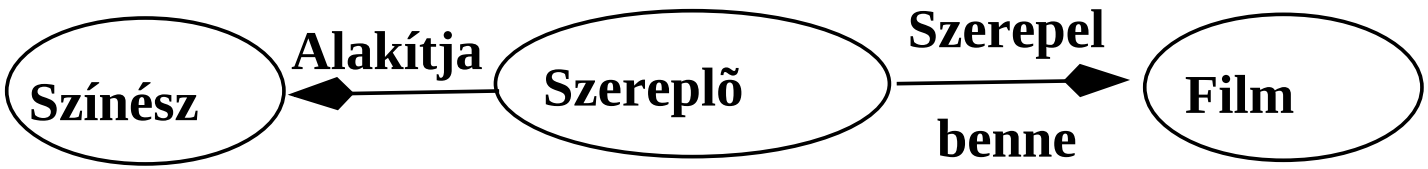
- Hierarchikus adatmodell
  Jó ott, ahol a reprezentálandó adatokban valódi hierarchia van, például biztosítós példa:



Adatnyilvántartás: fában, ami a hierarchiát tükrözi, a gyökér szerint rendezetten tárolva $\Longrightarrow$ a lekérdezés és módosítás, illetve az adatok elérése csak a fa ismeretében lehetséges.

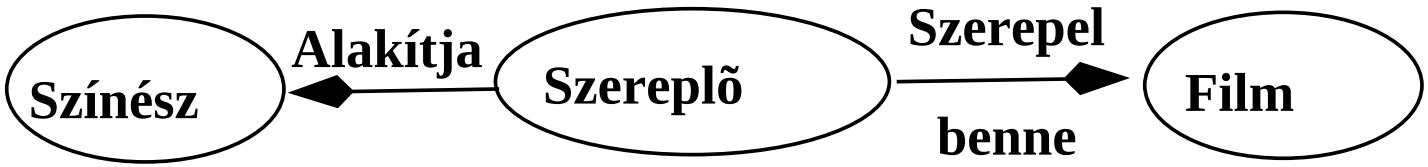
Korai modellek

- Hálós modell Irányított gráffal adjuk meg az adatok közötti logikai összefüggéseket, a csúcsok a rekordtípusok, a nyilak a kapcsolatok.



Korai modellek

- Hálós modell Irányított gráffal adjuk meg az adatok közötti logikai összefüggéseket, a csúcsok a rekordtípusok, a nyilak a kapcsolatok.



Mindkét modell hátránya: nincs magas szintű lekérdezés, bármilyen hozzáféréshez a tárolás pontos ismerete szükséges

Relációs adatmodell

- Jelenleg a legelterjedtebb modell
- E.F. Codd 1970-es cikkén alapul
- Fő elv: az adatbázist alkossák táblák (relációk)

Relációs adatmodell

- Jelenleg a legelterjedtebb modell
- E.F. Codd 1970-es cikkén alapul
- Fő elv: az adatbázist alkossák táblák (relációk)
- Előnye a hierarchikus és hálós modellel szemben:
  - magas szintű lekérdezés, a tárolási struktúra ismerete nélkül
  - jól átlátható, mégis pontos, elméleti háttere is van
  - a relációk mögött lehet bonyolult adatszerkezet is, de azt nem kell ismerni a működtetéshez

Ízelítő

Táblázat, reláció = fogalmi keret, egy-egy sor = egy-egy tárolandó adategyüttes.
Termelő(név, cím, termék, ár) tábla esetén:

név	**cím**	**termék**	**ár**
X. Kft	Sümeg	Kinder−tojás	127 Ft
.....	.....	.....	.....

Ízelítő

Táblázat, reláció = fogalmi keret, egy-egy sor = egy-egy tárolandó adategyüttes.
Termelő(név, cím, termék, ár) tábla esetén:

név	**cím**	**termék**	**ár**
X. Kft	Sümeg	Kinder−tojás	127 Ft
.....	.....	.....	.....

Lekérdezés: egyszerű, de hatékony, nem kell ismerni, hogy mi hogyan tárolódik. Pl. SQL-ben egy lekérdezés:
SELECT ár, név FROM termelő
WHERE termék=”Zizi” .5cm
Ez megkeresi az összes olyan (ár, név) párt, ami a „Zizi”-hez tartozik.

Jelenlegi rendszerek jellemzői

- főleg relációs modell, modellezésre pedig E/K diagram

Jelenlegi rendszerek jellemzői

- főleg relációs modell, modellezésre pedig E/K diagram
- egyre kisebb rendszerek (DBMS-ek PC-re)

Jelenlegi rendszerek jellemzői

- főleg relációs modell, modellezésre pedig E/K diagram
- egyre kisebb rendszerek (DBMS-ek PC-re)
- nagy adatbázisok (egyre hosszabb idejű tárolás, illetve képek, hangok, multimédiás cuccok) $\Longrightarrow$ harmadlagos tárolás CD-n

Jelenlegi rendszerek jellemzői

- főleg relációs modell, modellezésre pedig E/K diagram
- egyre kisebb rendszerek (DBMS-ek PC-re)
- nagy adatbázisok (egyre hosszabb idejű tárolás, illetve képek, hangok, multimédiás cuccok) $\Longrightarrow$ harmadlagos tárolás CD-n
- párhuzamos feldolgozás

Jövőbeni technológiák (részben már létezők)

- *objektumos adatbázisrendszerek*: ODL-es tervezés, szokásos objektumos megközelítés, összetett típusok (jól leírják a modellezni kívánt világot)

Jövőbeni technológiák (részben már létezők)

- *objektumos adatbázisrendszerek*: ODL-es tervezés, szokásos objektumos megközelítés, összetett típusok (jól leírják a modellezni kívánt világot)
- *megszorítások, triggerek*: aktív elemek (ha valami feltétel teljesül $\Longrightarrow$ beindul valami folyamat a rendszerben).
  - megszorítások: előre megadott feltételeknek mindig teljesülniük kell. Ha valamelyik sérülne: cselekvés,pl. letiltás.
  - triggerek: kódrészlet, ha valami adott helyzet bekövetkezik, akkor automatikusan kiváltódik valami esemény.

Jövőbeni technológiák (részben már létezők)

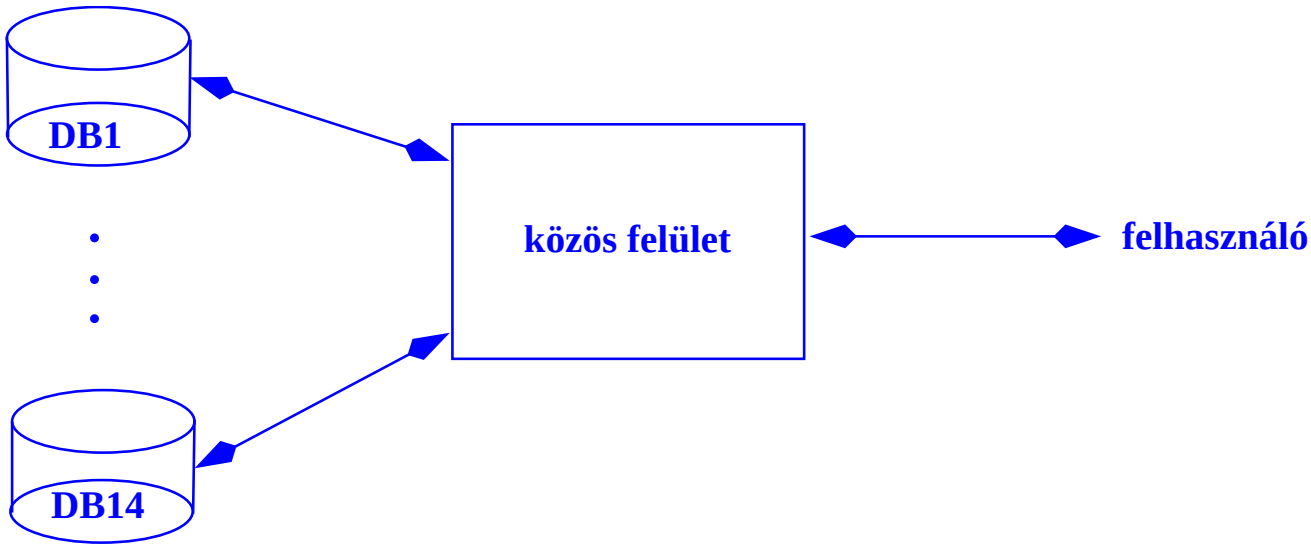
- *objektumos adatbázisrendszerek*: ODL-es tervezés, szokásos objektumos megközelítés, összetett típusok (jól leírják a modellezni kívánt világot)
- *megszorítások, triggerek*: aktív elemek (ha valami feltétel teljesül $\Longrightarrow$ beindul valami folyamat a rendszerben).
  - megszorítások: előre megadott feltételeknek mindig teljesülniük kell. Ha valamelyik sérülne: cselekvés,pl. letiltás.
  - triggerek: kódrészlet, ha valami adott helyzet bekövetkezik, akkor automatikusan kiváltódik valami esemény.
- *multimédiás adatok*: kép, hang, szöveg
  - sokkal nagyobb adatok
  - egyszerűbb műveletek is nehezek (pl. összehasonlítás), illetve új műveletek megjelenése
  - továbbítás problémája (nem egyszerre, hanem adagokban)

Jövőbeni technológiák (részben már létezők)

- *objektumos adatbázisrendszerek*: ODL-es tervezés, szokásos objektumos megközelítés, összetett típusok (jól leírják a modellezni kívánt világot)
- *megszorítások, triggerek*: aktív elemek (ha valami feltétel teljesül $\Longrightarrow$ beindul valami folyamat a rendszerben).
  - megszorítások: előre megadott feltételeknek mindig teljesülniük kell. Ha valamelyik sérülne: cselekvés,pl. letiltás.
  - triggerek: kódrészlet, ha valami adott helyzet bekövetkezik, akkor automatikusan kiváltódik valami esemény.
- *multimédiás adatok*: kép, hang, szöveg
  - sokkal nagyobb adatok
  - egyszerűbb műveletek is nehezek (pl. összehasonlítás), illetve új műveletek megjelenése
  - továbbítás problémája (nem egyszerre, hanem adagokban)

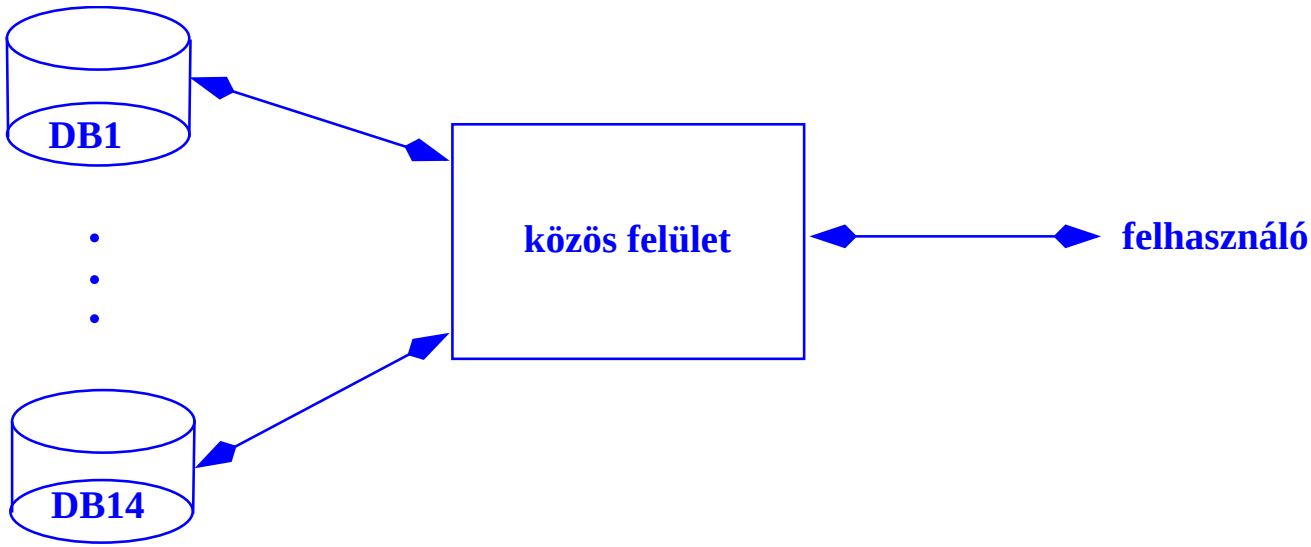
Jövőbeni technológiák (részben már létezők)

- *adattárházak*: cél az adathalmazok egységesítése. Sokféle adat, sok helyen, ugyanolyan vagy hasonló dolgokról, de különféle tárolási struktúrában. Egységesen akarjuk látni az adatokat (webes katalógus, egységes vállalati nyilvántartás).
  Megoldás az adattárház: átalakított, különböző DB-kből származó adatok „közös nevezőre” hozása.
  Nem kell lecserélni a kis adatbázisokat, hanem csak föléjük építünk egy struktúrát:



Jövőbeni technológiák (részben már létezők)

- *adattárházak*: cél az adathalmazok egységesítése. Sokféle adat, sok helyen, ugyanolyan vagy hasonló dolgokról, de különféle tárolási struktúrában. Egységesen akarjuk látni az adatokat (webes katalógus, egységes vállalati nyilvántartás).
  Megoldás az adattárház: átalakított, különböző DB-kből származó adatok „közös nevezőre” hozása.
  Nem kell lecserélni a kis adatbázisokat, hanem csak föléjük építünk egy struktúrát:



- *adatbányászat*: adatok között levő érdekes, szokatlan összefüggések keresése. Pl. aki fiatal férfi és . . . t vásárol, az vásárol . . . t is.

Adatmodellezés

- Célja: a modellezendő valóságdarabhoz adatbázisséma létrehozása.

Adatmodellezés

- Célja: a modellezendő valóságdarabhoz adatbázisséma létrehozása.
- Elvárás: jól írja le a valóságot, könnyű legyen a gyakori kérdéseket és módosításokat megtenni

Adatmodellezés

- Célja: a modellezendő valóságdarabhoz adatbázisséma létrehozása.
- Elvárás: jól írja le a valóságot, könnyű legyen a gyakori kérdéseket és módosításokat megtenni
- Részei:
  1. Terv készítése (nagyon fontos rész, ha rossz tervet csinálunk, később nehéz módosítani) valamilyen modellező eszköz/nyelv segítségével (E/K diagram, ODL-es megadás).

Adatmodellezés

- Célja: a modellezendő valóságdarabhoz adatbázisséma létrehozása.
- Elvárás: jól írja le a valóságot, könnyű legyen a gyakori kérdéseket és módosításokat megtenni
- Részei:
  1. Terv készítése (nagyon fontos rész, ha rossz tervet csinálunk, később nehéz módosítani) valamilyen modellező eszköz/nyelv segítségével (E/K diagram, ODL-es megadás).
  2. A terv átalakítása formálisabb leírássá (tipikusan E/K-ból relációs megadás).

Adatmodellezés

- Célja: a modellezendő valóságdarabhoz adatbázisséma létrehozása.
- Elvárás: jól írja le a valóságot, könnyű legyen a gyakori kérdéseket és módosításokat megtenni
- Részei:
  1. Terv készítése (nagyon fontos rész, ha rossz tervet csinálunk, később nehéz módosítani) valamilyen modellező eszköz/nyelv segítségével (E/K diagram, ODL-es megadás).
  2. A terv átalakítása formálisabb leírássá (tipikusan E/K-ból relációs megadás).
  3. Az adatbázisséma formális megadása a rendszer által kívánt DDL-en (ez az átalakítás már viszonylag automatikusan megy, a DDL persze rendszerfüggő).

Adatmodellezés

- Célja: a modellezendő valóságdarabhoz adatbázisséma létrehozása.
- Elvárás: jól írja le a valóságot, könnyű legyen a gyakori kérdéseket és módosításokat megtenni
- Részei:
  1. Terv készítése (nagyon fontos rész, ha rossz tervet csinálunk, később nehéz módosítani) valamilyen modellező eszköz/nyelv segítségével (E/K diagram, ODL-es megadás).
  2. A terv átalakítása formálisabb leírássá (tipikusan E/K-ból relációs megadás).
  3. Az adatbázisséma formális megadása a rendszer által kívánt DDL-en (ez az átalakítás már viszonylag automatikusan megy, a DDL persze rendszerfüggő).

Mi most az első lépéssel foglalkozunk, a tervezéssel, később lesz majd még arról szó, hogy hogyan kell a tervet átírni relációs sémára, aztán pedig az SQL DDL-jére.

Adatmodellező eszközök

Egy adatmodellező eszköz egy többé-kevésbé formális jelölésrendszer, adatok és a köztük levő kapcsolatok megadására. (ODL inkább formális, E/K kevésbé).

Adatmodellező eszközök

Egy adatmodellező eszköz egy többé-kevésbé formális jelölésrendszer, adatok és a köztük levő kapcsolatok megadására. (ODL inkább formális, E/K kevésbé).
Alapfogalmak:

- *adatok*, pl. pilóta, utas, járat

Adatmodellező eszközök

Egy adatmodellező eszköz egy többé-kevésbé formális jelölésrendszer, adatok és a köztük levő kapcsolatok megadására. (ODL inkább formális, E/K kevésbé).
Alapfogalmak:

- *adatok*, pl. pilóta, utas, járat
- *kapcsolatok*, pl. járat utasai, személyzete

Adatmodellező eszközök

Egy adatmodellező eszköz egy többé-kevésbé formális jelölésrendszer, adatok és a köztük levő kapcsolatok megadására. (ODL inkább formális, E/K kevésbé).
Alapfogalmak:

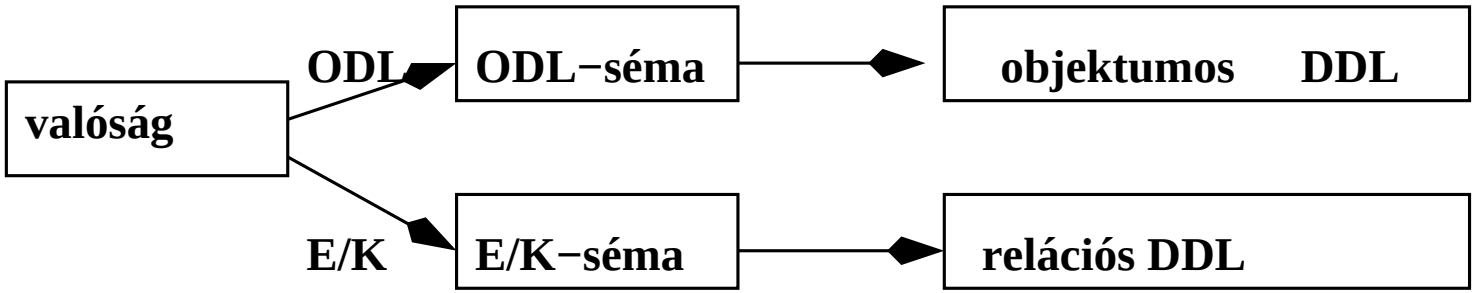
- *adatok*, pl. pilóta, utas, járat
- *kapcsolatok*, pl. járat utasai, személyzete
- *műveletek,* már ahol van, vannak modellek, amiknek vannak saját műveleteik, amiket könnyű megvalósítani.

Adatmodellező eszközök

Egy adatmodellező eszköz egy többé-kevésbé formális jelölésrendszer, adatok és a köztük levő kapcsolatok megadására. (ODL inkább formális, E/K kevésbé).
Alapfogalmak:

- *adatok*, pl. pilóta, utas, járat
- *kapcsolatok*, pl. járat utasai, személyzete
- *műveletek,* már ahol van, vannak modellek, amiknek vannak saját műveleteik, amiket könnyű megvalósítani.

Tipikus használat:



Az E/K-relációs séma-relációs DDL út a hagyományosabb.

ODL alapelvei

Cél: objektumos szemléletű DB tervezése, az adatbázis struktúrájának megadása objektumos terminológiával. CORBA része, objektumos programozási nyelvekhez jól passzol. Az ODL-es tervet könnyű objektumos DDL-be transzformálni (relációsra viszont nehézkes).

ODL alapelvei

Cél: objektumos szemléletű DB tervezése, az adatbázis struktúrájának megadása objektumos terminológiával. CORBA része, objektumos programozási nyelvekhez jól passzol. Az ODL-es tervet könnyű objektumos DDL-be transzformálni (relációsra viszont nehézkes).
Alapelvek:

- A világot objektumokkal írjuk le (objektum = megfogható, megkülönböztethető egyed, pl. egy-egy járat, utas, dolgozó).

ODL alapelvei

Cél: objektumos szemléletű DB tervezése, az adatbázis struktúrájának megadása objektumos terminológiával. CORBA része, objektumos programozási nyelvekhez jól passzol. Az ODL-es tervet könnyű objektumos DDL-be transzformálni (relációsra viszont nehézkes).
Alapelvek:

- A világot objektumokkal írjuk le (objektum = megfogható, megkülönböztethető egyed, pl. egy-egy járat, utas, dolgozó).
- Minden objektumnak egyedi azonosítója van (OID), ez automatikusan generálódik neki és minden más OID-től különböző.

ODL alapelvei

Cél: objektumos szemléletű DB tervezése, az adatbázis struktúrájának megadása objektumos terminológiával. CORBA része, objektumos programozási nyelvekhez jól passzol. Az ODL-es tervet könnyű objektumos DDL-be transzformálni (relációsra viszont nehézkes).
Alapelvek:

- A világot objektumokkal írjuk le (objektum = megfogható, megkülönböztethető egyed, pl. egy-egy járat, utas, dolgozó).
- Minden objektumnak egyedi azonosítója van (OID), ez automatikusan generálódik neki és minden más OID-től különböző.
- Az objektumokat osztályokba soroljuk, az osztály elemei hasonlóak, ugyanolyan dolgokat tartunk róluk nyilván (pl. egy osztály lehet az összes utas, összes járat). Az egyes értékek persze lehetnek mások (az utasok neve különbözik, de minden utasnak van neve). Egy objektumot általában egy rekorddal adunk meg, az egyes mezők a nyilvántartott tulajdonságoknak felelnek meg.

Osztálydeklaráció

- Meg kell adni az osztály *nevét*.

Osztálydeklaráció

- Meg kell adni az osztály *nevét*.
- Az osztályhoz tartozó *attribútumok*: az osztályba tartozó objektumok jellemzői, lehetőleg egyszerűbb adattípusokkal megadva. (Erről majd később.)

Osztálydeklaráció

- Meg kell adni az osztály *nevét*.
- Az osztályhoz tartozó *attribútumok*: az osztályba tartozó objektumok jellemzői, lehetőleg egyszerűbb adattípusokkal megadva. (Erről majd később.)
- *Kapcsolatok* az osztályok között, ezeknek is van típusa, aszerint, hogy egy objektum egy másik osztály egy vagy pedig több objektumával kapcsolódik-e össze (pl. egy járatnak egy kapitánya van, de sok utasa).

Osztálydeklaráció

- Meg kell adni az osztály *nevét*.
- Az osztályhoz tartozó *attribútumok*: az osztályba tartozó objektumok jellemzői, lehetőleg egyszerűbb adattípusokkal megadva. (Erről majd később.)
- *Kapcsolatok* az osztályok között, ezeknek is van típusa, aszerint, hogy egy objektum egy másik osztály egy vagy pedig több objektumával kapcsolódik-e össze (pl. egy járatnak egy kapitánya van, de sok utasa).

Az osztálydeklaráció formája

interface <osztály neve> {<attribútumok listája, kapcsolatok>};

Példa

```
interface Film {
        attribute string cím;
        attribute int hossz;
        attribute int év;
        attribute enum Szalag{színes, fekete-fehér} szalagfajta;
            };
```

Az osztály neve Film, négy attribútuma van. Az attribute kulcsszó után megadjuk az attribútum típusát (a lehetséges típusokról később), majd az attribútum nevét. Az utolsó sorban egy felsorolás jellegű (enum), szalagfajta nevű attribútumot definiálunk, ami a Szalag (kételemű) halmazból veszi az értékét.

Példa

```
interface Film {
        attribute string cím;
        attribute int hossz;
        attribute int év;
        attribute enum Szalag{színes, fekete-fehér} szalagfajta;
            };
```

Az osztály neve Film, négy attribútuma van. Az attribute kulcsszó után megadjuk az attribútum típusát (a lehetséges típusokról később), majd az attribútum nevét. Az utolsó sorban egy felsorolás jellegű (enum), szalagfajta nevű attribútumot definiálunk, ami a Szalag (kételemű) halmazból veszi az értékét.

Ez persze csak a kezdete egy osztálydeklarációnak, kapcsolatokat még nem is adtunk meg.

Egy objektum egy rekordnak felel meg, pl. a fenti megadás szerint a Film osztály egy objektuma pl. (Amélie csodálatos élete, 120, 2000, színes).

Még egy példa

```
interface Színész {
        attribute string név;
        attribute Struct Cím{string város, string utca} lakcím;
            };
```

Itt a második attribútum struktúra típusú, ami két mezőből áll, az első mező neve város, típusa string, a másodiké utca, típusa string. Az attribútum neve lakcím.

Kapcsolatok megadása

Az objektumok tulajdonságait az attribútumokkal adjuk meg, az objektumok közötti hivatkozásokat pedig a kapcsolatokkal. Egy objektum kapcsolódhat egy vagy több másik objektumhoz is.

Kapcsolatok megadása

Az objektumok tulajdonságait az attribútumokkal adjuk meg, az objektumok közötti hivatkozásokat pedig a kapcsolatokkal. Egy objektum kapcsolódhat egy vagy több másik objektumhoz is.A kapcsolatokat ugyanott írjuk le, ahol az attribútumokat, a megadás módja:
relationship <osztálynév> <kapcsolatnév>;
ha egy objektumhoz vezet a kapcsolat, illetve
relationship <kollekcióoperátor>< <osztálynév> > <kapcsolatnév>;
ha több (a kollekcióoperátor mondja meg, hogy milyen) objektumhoz vezet a kapcsolat.
A lehetséges kollekcióoperátorokról (Set, Bag, List, Array) majd később.

Példa kapcsolat megadására

A Film osztályba
relationship Set<Színész> szereplők;
és
relationship Színész főszereplő; kell.
Az első esetben egy filmhez a színészek egy halmaza kapcsolódik, a második esetben egy filmhez egy darab színész tartozik.

Példa kapcsolat megadására

A Film osztályba
relationship Set<Színész> szereplők;
és
relationship Színész főszereplő; kell.
Az első esetben egy filmhez a színészek egy halmaza kapcsolódik, a második esetben egy filmhez egy darab színész tartozik.
Fontos! A kapcsolatot a másik osztálynál is jelölni kell és meg kell adni, hogy melyik kapcsolat inverzéről van szó.

Így a Színész osztályba is kell

```
relationship Set<Film> szerepelBenne;
      inverse Film::szereplők;
```

és

```
relationship Set<Film> főszereplőBenne;
      inverse Film::főszereplő;
```

Így a Színész osztályba is kell

```
relationship Set<Film> szerepelBenne;
      inverse Film::szereplők;
```

és

```
relationship Set<Film> főszereplőBenne;
      inverse Film::főszereplő;
```

És persze a Film osztályba is kell a két inverse:

```
relationship Set<Színész> szereplők;
      inverse Színész::szerepelBenne;
```

és

```
relationship  Színész főszereplő;
      inverse Színész::főszereplőBenne;
```

Inverzek

Itt persze ugyanazon dolog két nézetéről van szó. Fontos konzisztenciatényező az inverz-párok feltüntetése, mert -.5cm

- Elvárjuk, hogy ha X.Y. szerepel egy filmnél, mint szereplő, akkor az a film szerepeljen nála a szereplBenne kapcsolatnál.
- Általában azok a jól megfogott kapcsolatok, amikhez könnyű, természetes inverzet találni.

Igazából egy dolog van csak, egy ilyen fajta megfeleltetés:

Színész	Film
A. Tautou	Amélie csodálatos élete
A. Tautou	Szeretni bolondulásig
M. Kasovitz	Amélie csodálatos élete
M. Kasovitz	Férfiek mélyrepülésben

Ennek kétféle elérése a két kapcsolat.

Kapcsolatok jellege

Egy C és egy D osztály közötti kapcsolat lehet

- több-több (sok-sok, N:N) kapcsolat: egy C-beli objektumhoz több D-beli és egy D-belihez több C-beli is tartozhat (pl. a szereplők/szerepelBenne kapcsolatpár).

Kapcsolatok jellege

Egy C és egy D osztály közötti kapcsolat lehet

- több-több (sok-sok, N:N) kapcsolat: egy C-beli objektumhoz több D-beli és egy D-belihez több C-beli is tartozhat (pl. a szereplők/szerepelBenne kapcsolatpár).
- több-egy (sok-egy, N:1) kapcsolat: egy C-belihez csak egy D-beli tartozhat, de egy D-belihez tartozhat több C-beli is (pl. a Film és a Színész osztályok között levő főszereplője/főszereplőBenne pár).

Kapcsolatok jellege

Egy C és egy D osztály közötti kapcsolat lehet

- több-több (sok-sok, N:N) kapcsolat: egy C-beli objektumhoz több D-beli és egy D-belihez több C-beli is tartozhat (pl. a szereplők/szerepelBenne kapcsolatpár).
- több-egy (sok-egy, N:1) kapcsolat: egy C-belihez csak egy D-beli tartozhat, de egy D-belihez tartozhat több C-beli is (pl. a Film és a Színész osztályok között levő főszereplője/főszereplőBenne pár).
- egy-egy (1:1) kapcsolat: egy C-belihez csak egy D-beli és egy D-belihez csak egy C-beli tartozhat (férj-feleség kapcsolat pl.).

Kapcsolatok jellege

Egy C és egy D osztály közötti kapcsolat lehet

- több-több (sok-sok, N:N) kapcsolat: egy C-beli objektumhoz több D-beli és egy D-belihez több C-beli is tartozhat (pl. a szereplők/szerepelBenne kapcsolatpár).
- több-egy (sok-egy, N:1) kapcsolat: egy C-belihez csak egy D-beli tartozhat, de egy D-belihez tartozhat több C-beli is (pl. a Film és a Színész osztályok között levő főszereplője/főszereplőBenne pár).
- egy-egy (1:1) kapcsolat: egy C-belihez csak egy D-beli és egy D-belihez csak egy C-beli tartozhat (férj-feleség kapcsolat pl.).

A kapcsolat jellege azt mutatja, mennyire függvényszerű a kapcsolat az objektumok között. A kapcsolat jellege deklarációs kérdés, az osztály megadásakor döntjük el (azzal, hogy használunk-e kollekcióoperátort vagy sem).
(Egy több-több kapcsolat esetén is előfordulhat persze, hogy egy adott objektum csak egy másikhoz csatlakozik.)

Típusok az ODL-ben

Vannak alaptípusok, építkezési lehetőségek és megszorítások, amik szabályozzák az építkezést.

Alaptípusok

- Atomi típusok (elemi típusok): integer, real, float, char, string, boolean, enum

Típusok az ODL-ben

Vannak alaptípusok, építkezési lehetőségek és megszorítások, amik szabályozzák az építkezést.

Alaptípusok

- Atomi típusok (elemi típusok): integer, real, float, char, string, boolean, enum
- Interface típusok: mi magunk csináljuk őket, a deklarált osztályok ezek (pl. Film, Színész)

Típusok az ODL-ben

Vannak alaptípusok, építkezési lehetőségek és megszorítások, amik szabályozzák az építkezést.

Alaptípusok

- Atomi típusok (elemi típusok): integer, real, float, char, string, boolean, enum
- Interface típusok: mi magunk csináljuk őket, a deklarált osztályok ezek (pl. Film, Színész)

Típusok az ODL-ben

Típuskonstruktorok

- *Halmaz*: ha T egy típus, akkor Set$< T >$ a T típusú elemek halmaza

Típusok az ODL-ben

Típuskonstruktorok

- *Halmaz*: ha T egy típus, akkor Set$< T >$ a T típusú elemek halmaza
- *Multihalmaz*: ha T egy típus, akkor Bag$< T >$ a T típusú elemek multihalmaza, azaz egy elem többször is szerepelhet

Típusok az ODL-ben

Típuskonstruktorok

- *Halmaz*: ha T egy típus, akkor Set$< T >$ a T típusú elemek halmaza
- *Multihalmaz*: ha T egy típus, akkor Bag$< T >$ a T típusú elemek multihalmaza, azaz egy elem többször is szerepelhet
- *Lista*: ha T egy típus, akkor List$< T >$ a T típusú elemek listája, pl. string=List$<$ *char* $>$

Típusok az ODL-ben

Típuskonstruktorok

- *Halmaz*: ha T egy típus, akkor Set$< T >$ a T típusú elemek halmaza
- *Multihalmaz*: ha T egy típus, akkor Bag$< T >$ a T típusú elemek multihalmaza, azaz egy elem többször is szerepelhet
- *Lista*: ha T egy típus, akkor List$< T >$ a T típusú elemek listája, pl. string=List$<$ *char* $>$
- *Tömb*: ha T egy típus, akkor Array$< T, i >$ a T típusú elemek i hosszú tömbje, pl. Array$<$ char, 12 $>=$ 12 hosszú karakterlánc

Típusok az ODL-ben

Típuskonstruktorok

- *Halmaz*: ha T egy típus, akkor Set$< T >$ a T típusú elemek halmaza
- *Multihalmaz*: ha T egy típus, akkor Bag$< T >$ a T típusú elemek multihalmaza, azaz egy elem többször is szerepelhet
- *Lista*: ha T egy típus, akkor List$< T >$ a T típusú elemek listája, pl. string=List$<$ *char* $>$
- *Tömb*: ha T egy típus, akkor Array$< T, i >$ a T típusú elemek i hosszú tömbje, pl. Array$<$ char, 12 $>=$ 12 hosszú karakterlánc
- *Struktúra*: ha $T_1, T_2, \ldots, T_n$ típusok, $f_1, f_2, \ldots, f_n$ pedig mezőnevek, akkor Struct $<$ Név $>$ $\{T_1 f_1, T_2 f_2, \ldots, T_n f_n\}$ n mezőből álló $<$ Név $>$ nevű struktúra, ahol a mezők nevei $f_1, f_2, \ldots, f_n$, típusai pedig $T_1, T_2, \ldots, T_n$.
  Például: Struct Cím{string város, string utca}

Az első négy (Set, Bag, List, Array) típuskonstruktort **kollekcióoperátornak** hívjuk.

Típusok az ODL-ben

Típuskonstruktorok

- *Halmaz*: ha T egy típus, akkor Set$< T >$ a T típusú elemek halmaza
- *Multihalmaz*: ha T egy típus, akkor Bag$< T >$ a T típusú elemek multihalmaza, azaz egy elem többször is szerepelhet
- *Lista*: ha T egy típus, akkor List$< T >$ a T típusú elemek listája, pl. string=List$<$ *char* $>$
- *Tömb*: ha T egy típus, akkor Array$< T, i >$ a T típusú elemek i hosszú tömbje, pl. Array$<$ char, 12 $>=$ 12 hosszú karakterlánc
- *Struktúra*: ha $T_1, T_2, \ldots, T_n$ típusok, $f_1, f_2, \ldots, f_n$ pedig mezőnevek, akkor Struct $<$ Név $>$ $\{T_1 f_1, T_2 f_2, \ldots, T_n f_n\}$ n mezőből álló $<$ Név $>$ nevű struktúra, ahol a mezők nevei $f_1, f_2, \ldots, f_n$, típusai pedig $T_1, T_2, \ldots, T_n$.
  Például: Struct Cím{string város, string utca}

Az első négy (Set, Bag, List, Array) típuskonstruktort **kollekcióoperátornak** hívjuk.

Megkötések

- *Attribútum típusa*: lehet atomi típus, struktúra atomi típusú mezőkkel, illetve ezekre lehet még egy kollekció operátort vagy egy struktúrát rakni (de csak egyszer!!!!) (Ezzel elég bonyolult típusokat lehet csinálni, de önmérséklet, mert nehéz lesz megvalósítani, ha túl bonyolult).

Típusok az ODL-ben

Megkötések

- *Attribútum típusa*: lehet atomi típus, struktúra atomi típusú mezőkkel, illetve ezekre lehet még egy kollekció operátort vagy egy struktúrát rakni (de csak egyszer!!!!) (Ezzel elég bonyolult típusokat lehet csinálni, de önmérséklet, mert nehéz lesz megvalósítani, ha túl bonyolult).
- *Kapcsolat típusa*: interface típus vagy interface típusra egyszer alkalmazott kollekcióoperátor (struktúra nem lehet!!!)

Megjegyzések

- Ugyanaz a típus nem lehet attribútum és kapcsolat típusa is.

Típusok az ODL-ben

Megkötések

- *Attribútum típusa*: lehet atomi típus, struktúra atomi típusú mezőkkel, illetve ezekre lehet még egy kollekció operátort vagy egy struktúrát rakni (de csak egyszer!!!!) (Ezzel elég bonyolult típusokat lehet csinálni, de önmérséklet, mert nehéz lesz megvalósítani, ha túl bonyolult).
- *Kapcsolat típusa*: interface típus vagy interface típusra egyszer alkalmazott kollekcióoperátor (struktúra nem lehet!!!)

Megjegyzések

- Ugyanaz a típus nem lehet attribútum és kapcsolat típusa is.
- Kollekcióoperátort mindkét helyen lehet használni, de amire alkalmazom az más (elemi típus, illetve interface).

Típusok az ODL-ben

Megkötések

- *Attribútum típusa*: lehet atomi típus, struktúra atomi típusú mezőkkel, illetve ezekre lehet még egy kollekció operátort vagy egy struktúrát rakni (de csak egyszer!!!!) (Ezzel elég bonyolult típusokat lehet csinálni, de önmérséklet, mert nehéz lesz megvalósítani, ha túl bonyolult).
- *Kapcsolat típusa*: interface típus vagy interface típusra egyszer alkalmazott kollekcióoperátor (struktúra nem lehet!!!)

Megjegyzések

- Ugyanaz a típus nem lehet attribútum és kapcsolat típusa is.
- Kollekcióoperátort mindkét helyen lehet használni, de amire alkalmazom az más (elemi típus, illetve interface).
- Példa:
  Array$<$ Struct $N\{\text{string } m_1, \text{string } m_2\}, 10 >$ lehet egy attribútum típusa

Adatbázisok elmélete

E/K modell

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

3. előadás

E/K diagram

Eddig azt néztük meg, hogy ODL-ben hogyan lehet osztályokat, kapcsolatokat megadni és ezzel a DB fogalmi keretét kialakítani.
Most egy másik módszer jön, az E/K diagram, ezt könnyen át lehet majd írni relációs sémára.

E/K diagram

Eddig azt néztük meg, hogy ODL-ben hogyan lehet osztályokat, kapcsolatokat megadni és ezzel a DB fogalmi keretét kialakítani.
Most egy másik módszer jön, az E/K diagram, ezt könnyen át lehet majd írni relációs sémára.
E/K= egyed-kapcsolat vagy entitás-relációs (E/R, entity-relationship) modell
Szemléletes, könnyű vele dolgozni. Egy rajzot készítünk, ez ábrázolja az adatelemeket és a köztük levő kapcsolatot is.

Alapfogalmak

Hasonlítanak az alapelemek az ODL-hez:

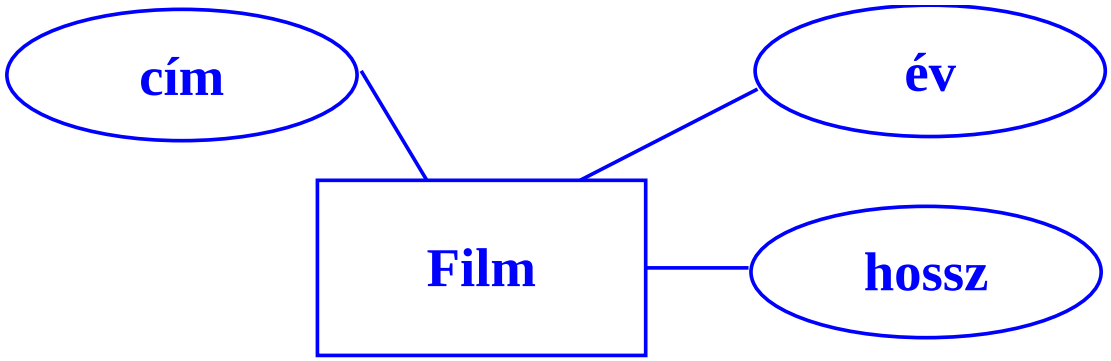
- *Egyedhalmaz* (kb. mint az osztály az ODL-ben): elemei az egyedek (ODL-es objektumok), de itt nincs egyedi azonosító, az egyedek az attribútumaikkal és a kapcsolataikkal azonosítódnak.
  Rajzon:

Alapfogalmak

Hasonlítanak az alapelemek az ODL-hez:

- *Egyedhalmaz* (kb. mint az osztály az ODL-ben): elemei az egyedek (ODL-es objektumok), de itt nincs egyedi azonosító, az egyedek az attribútumaikkal és a kapcsolataikkal azonosítódnak.
  Rajzon:

- *Attribútumok*: értékeik egy egyed tulajdonságait adják meg, mint az ODL-nél, de itt nincs formális előírás a típusokra, csak annyi, hogy legyenek egyszerűek, hogy könnyű legyen relációsra átírni.
  Szöveges jelölés: Film(Cím, Hossz, . . .), rajzon:

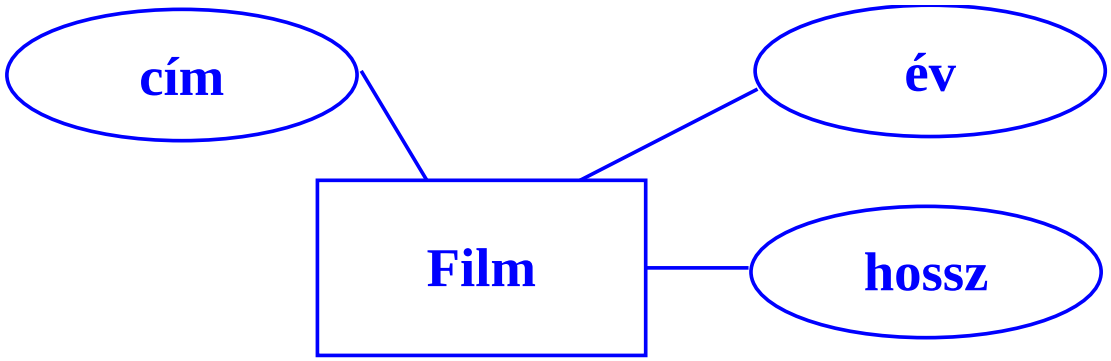


Alapfogalmak

Hasonlítanak az alapelemek az ODL-hez:

- *Egyedhalmaz* (kb. mint az osztály az ODL-ben): elemei az egyedek (ODL-es objektumok), de itt nincs egyedi azonosító, az egyedek az attribútumaikkal és a kapcsolataikkal azonosítódnak.
  Rajzon:

- *Attribútumok*: értékeik egy egyed tulajdonságait adják meg, mint az ODL-nél, de itt nincs formális előírás a típusokra, csak annyi, hogy legyenek egyszerűek, hogy könnyű legyen relációsra átírni.
  Szöveges jelölés: Film(Cím, Hossz, . . .), rajzon:



Alapfogalmak

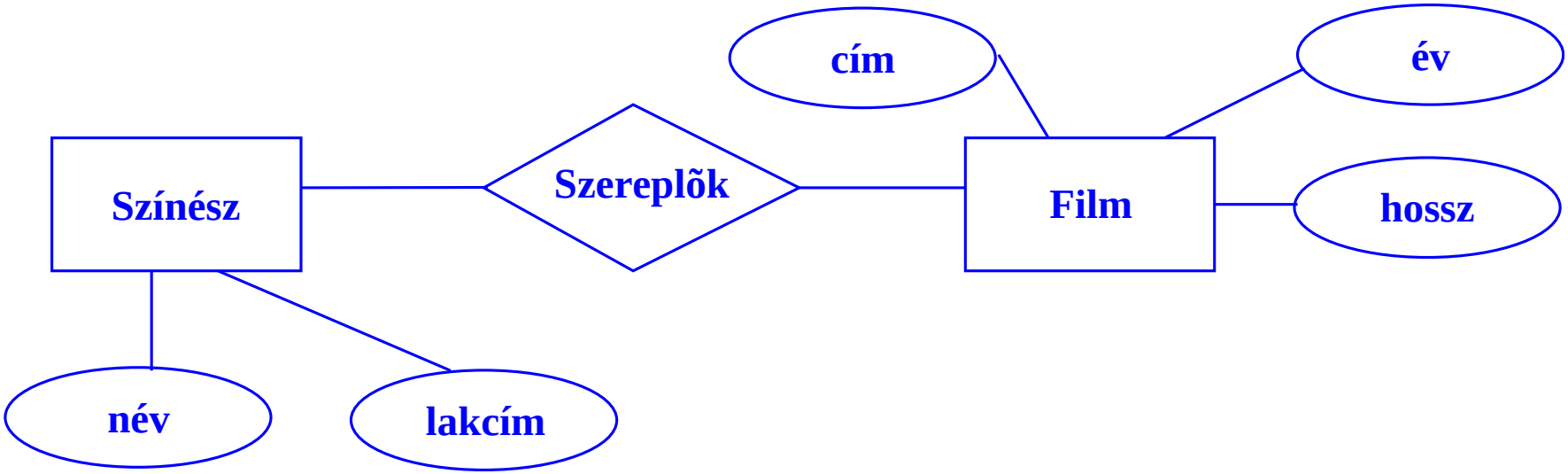
- *Kapcsolatok*: egyedhalmazok közötti viszony, máshogy van, mint ODL-ben.
- ODL-ben minden kapcsolatot mindkét irányban reprezentálunk, itt egy kapcsolat = egy vonal
- ODL-ben minden kapcsolat bináris (két osztály között megy), E/K-ban lehetnek többágú kapcsolatok is

Alapfogalmak

- *Kapcsolatok*: egyedhalmazok közötti viszony, máshogy van, mint ODL-ben.
- ODL-ben minden kapcsolatot mindkét irányban reprezentálunk, itt egy kapcsolat = egy vonal
- ODL-ben minden kapcsolat bináris (két osztály között megy), E/K-ban lehetnek többágú kapcsolatok is

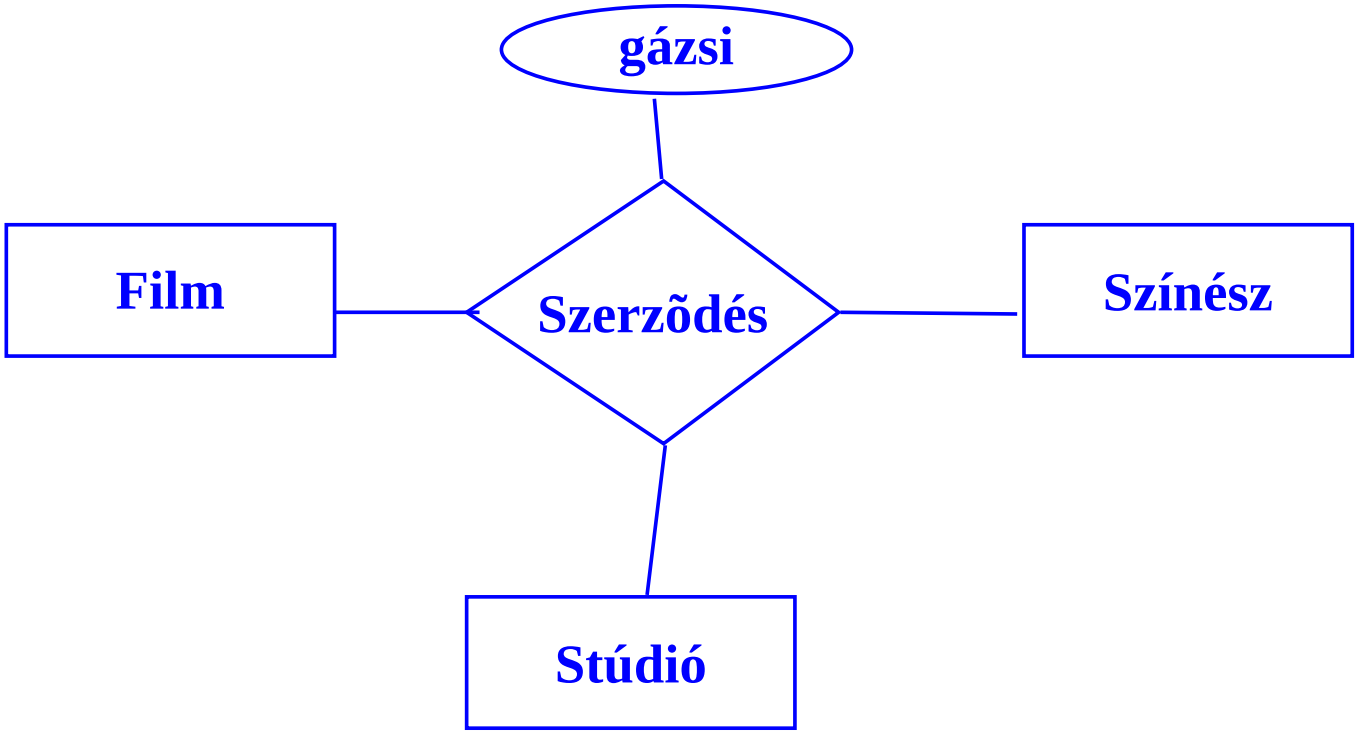
Alapfogalmak

Jelölés szövegesen: Szereplők(Film, Színész), illetve rajzon:



Ha az $R(E_1, E_2, \ldots, E_{10})$ kapcsolat 10 egyedhalmazt köt össze, akkor az R kapcsolat egy példánya egy 10 hosszú vektor $(e_1, e_2, \ldots, e_{10})$, ahol az e_i egy egyed az E_i egyedhalmazból.

Fontos különbség még az ODL-hez képest, hogy az E/K modellben a kapcsolatnak is lehet attribútuma:



Itt a gázsi a szerződéshez tartozik, ami a filmet, a színészt és a stúdiót köti össze. Lehetne úgy is csinálni, hogy a Szerződés kapcsolatnak lenne egy negyedik egyedhalmaza is, a Gázsi, egyetlen attribútummal, az összeggel, de felesleges olyan egyedhalmazt létrehozni, aminek csak egy attribútuma van.

Kapcsolatok típusa

A bináris kapcsolatoknál ugyanúgy van, mint az ODL-nél: van több-több, több-egy és egy-egy kapcsolat. A többágú kapcsolat egy kicsit bonyolultabb.

Bináris kapcsolat

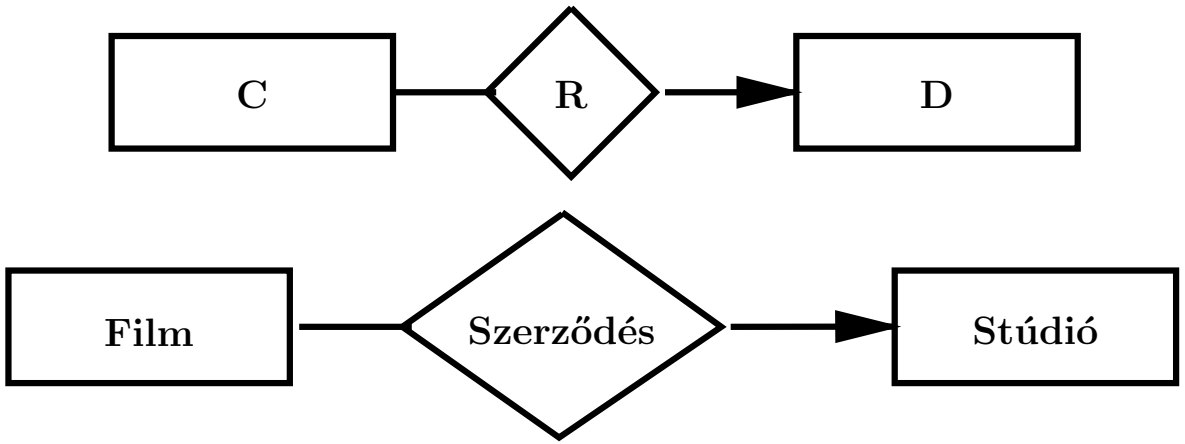
Az „egy” irányt nyíl jelzi, azaz ott van nyíl, amelyik egyedhalmazból csak egy tartozhat a másik egyedhalmaz egy egyedéhez.

Kapcsolatok típusa

A bináris kapcsolatoknál ugyanúgy van, mint az ODL-nél: van több-több, több-egy és egy-egy kapcsolat. A többágú kapcsolat egy kicsit bonyolultabb.

Bináris kapcsolat

Az „egy” irányt nyíl jelzi, azaz ott van nyíl, amelyik egyedhalmazból csak egy tartozhat a másik egyedhalmaz egy egyedéhez.
Például, ha a *C* egyedhalmaz egy egyedéhez csak egy *D*-beli tartozhat az *R* kapcsolatnál, de egy *D*-belihez tartozhat több *C*-beli is, akkor:



Ha egy-egy kapcsolat van, akkor persze mindkét oldalra kell a nyíl.

Többágú kapcsolat típusa

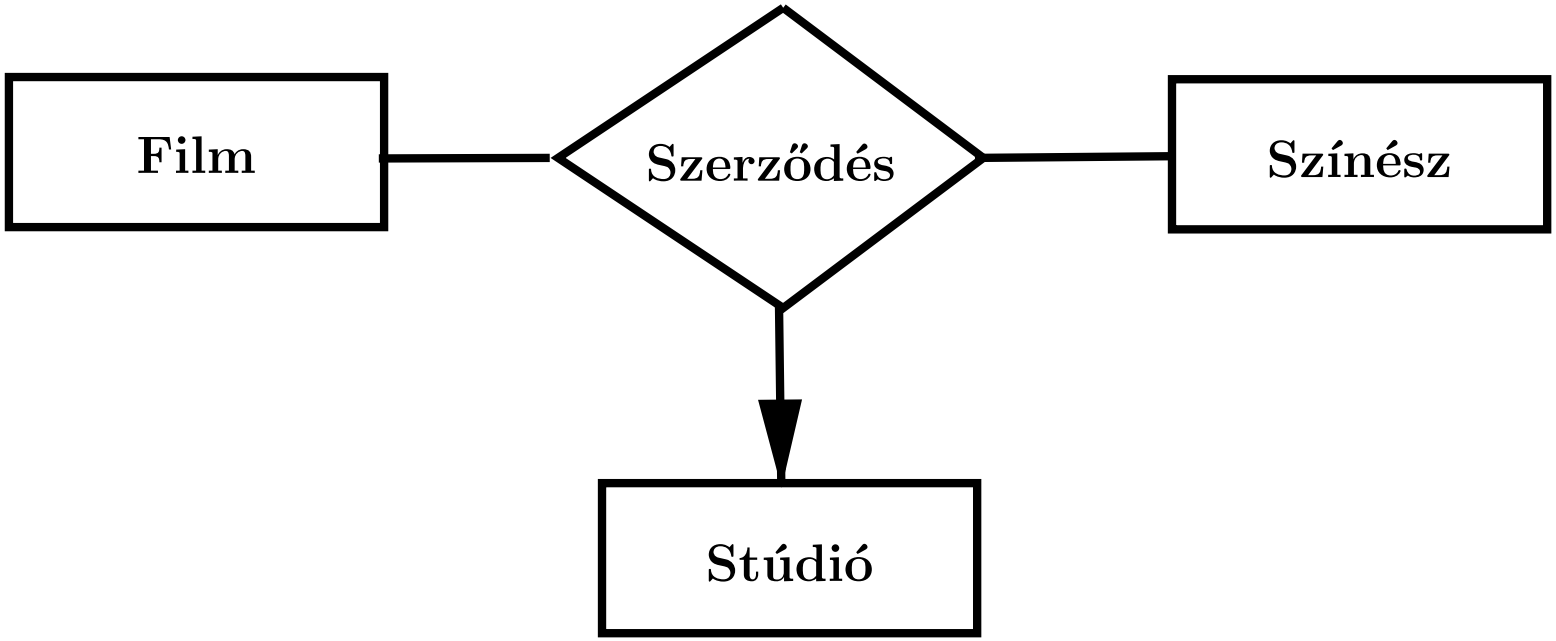
Az $R(E_1, E_2, \ldots, E_n)$ kapcsolat E_i felé egyirányú, ha igaz az, hogy a maradék $E_1, E_2, \ldots, E_{i-1}, E_{i+1}, \ldots, E_n$ egyedhalmazokból bárhogy választva ki egy-egy egyedet (e_k-t az E_k-ból), maximum egy olyan E_i-beli e_i egyed van, amivel az $(e_1, e_2, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_n)$ egyedvektorra az R kapcsolat fennáll.

Többágú kapcsolat típusa

Az $R(E_1, E_2, \ldots, E_n)$ kapcsolat E_i felé egyirányú, ha igaz az, hogy a maradék $E_1, E_2, \ldots, E_{i-1}, E_{i+1}, \ldots, E_n$ egyedhalmazokból bárhogy választva ki egy-egy egyedet (e_k-t az E_k-ból), maximum egy olyan E_i-beli e_i egyed van, amivel az $(e_1, e_2, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_n)$ egyedvektorra az R kapcsolat fennáll.
(Természetesen lehet egy többágú kapcsolat egynél több irányban is „egy” jellegű.)

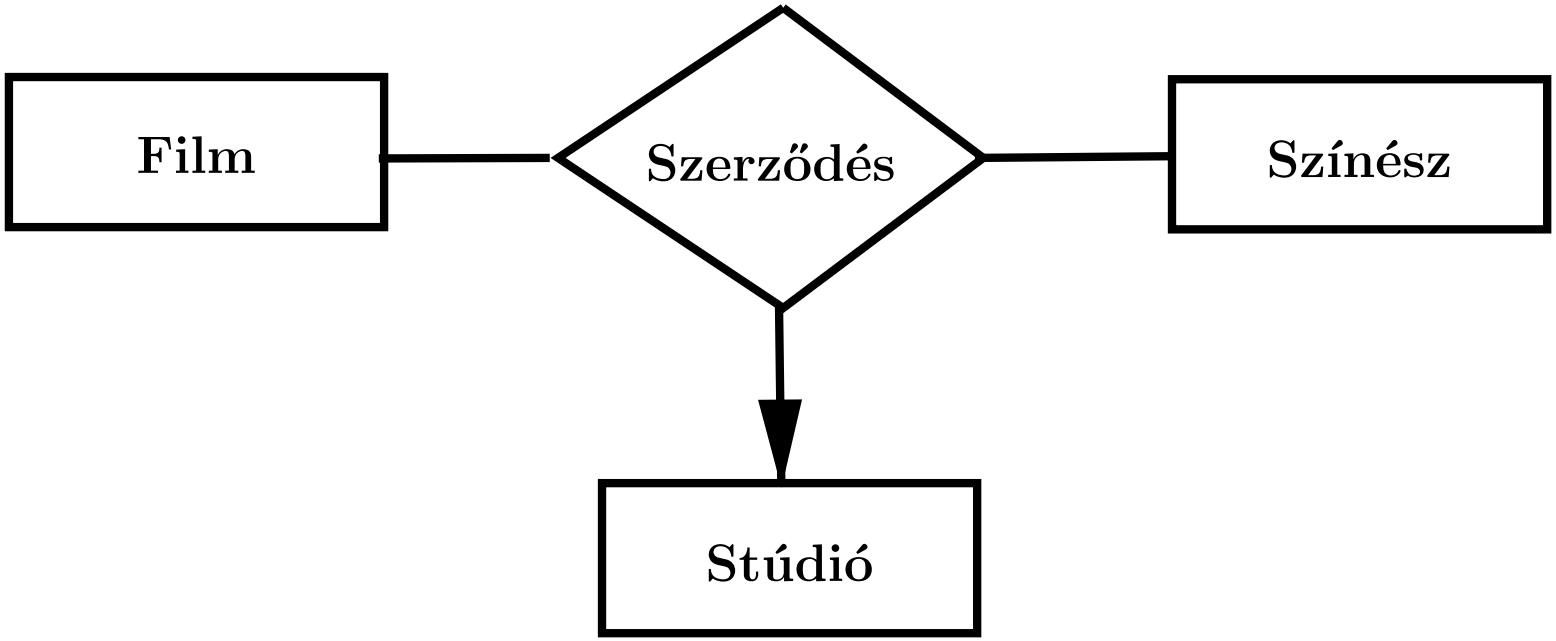
Többágú kapcsolat típusa

Példa:



Többágú kapcsolat típusa

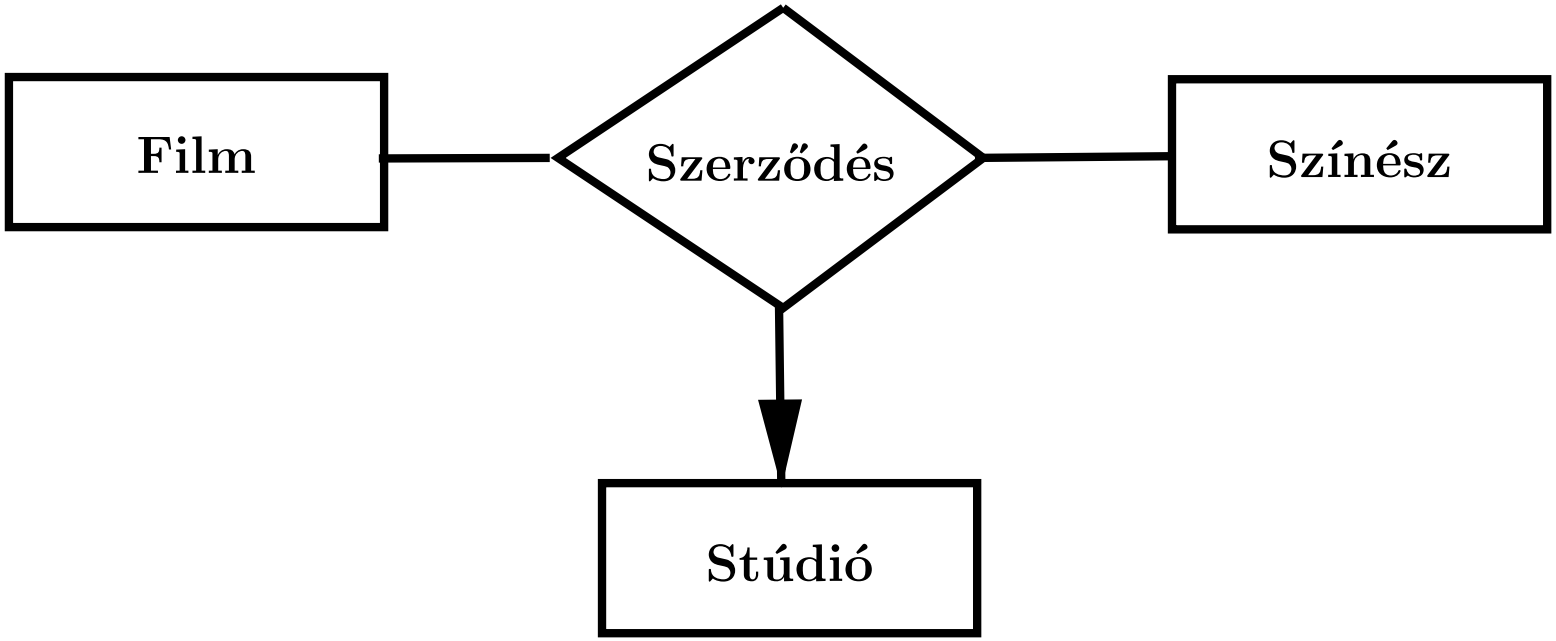
Példa:



Egy rögzített (film, színész) párhoz csak egy stúdió tartozik, de pl. egy rögzített (stúdió, film) párhoz tartozhat több színész.

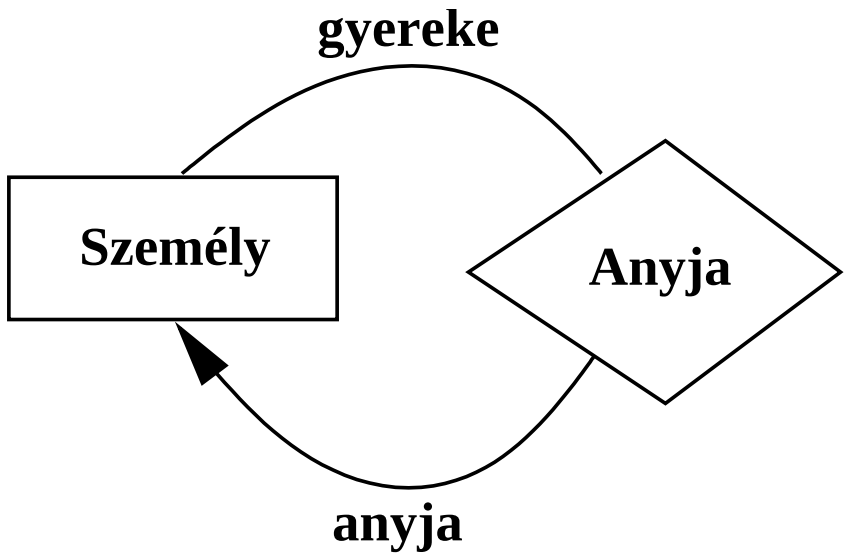
Többágú kapcsolat típusa

Példa:



Egy rögzített (film, színész) párhoz csak egy stúdió tartozik, de pl. egy rögzített (stúdió, film) párhoz tartozhat több színész.
Nem minden írható le ilyen módon, de nem baj, úgyis az a fontos, hogy majd a relációs megadásnál pontosak tudjunk lenni. A fenti példánál, a rajzon nem tudjuk jelölni azt, hogy már a film egyedül meghatározza a stúdiót, de a relációs modellben lesz erre eszközünk.

Ha egy kapcsolatban egy egyedhalmaz többször is szerepel, akkor a nyilakon/vonalakon jelöljük a különböző szerepeket. pl:



Fontos megcímkézni a vonalakat, mert ahhoz a szerephez tartozik az „egy” nyíl, ami az anyára vonatkozik, a gyerekághoz nem kell.

Sokágú kapcsolat átírása binárisra

Miért kell/jó ez?

- átláthatóbb a bináris

Sokágú kapcsolat átírása binárisra

Miért kell/jó ez?

- átláthatóbb a bináris
- a megvalósításban könnyebben kezelhető

Sokágú kapcsolat átírása binárisra

Miért kell/jó ez?

- átláthatóbb a bináris
- a megvalósításban könnyebben kezelhető
- látszik, hogy nem baj, hogy az ODL csak binárist tud, hisz a többágút is át lehet ilyenné írni

Sokágú kapcsolat átírása binárisra

Miért kell/jó ez?

- átláthatóbb a bináris
- a megvalósításban könnyebben kezelhető
- látszik, hogy nem baj, hogy az ODL csak binárist tud, hisz a többágút is át lehet ilyenné írni

Az átíráshoz fő ötlet: egy $R(E_1, E_2, \ldots, E_n)$ kapcsolat egy eleme egy n egyedből álló vektor: $(e_1, e_2, \ldots, e_n)$.

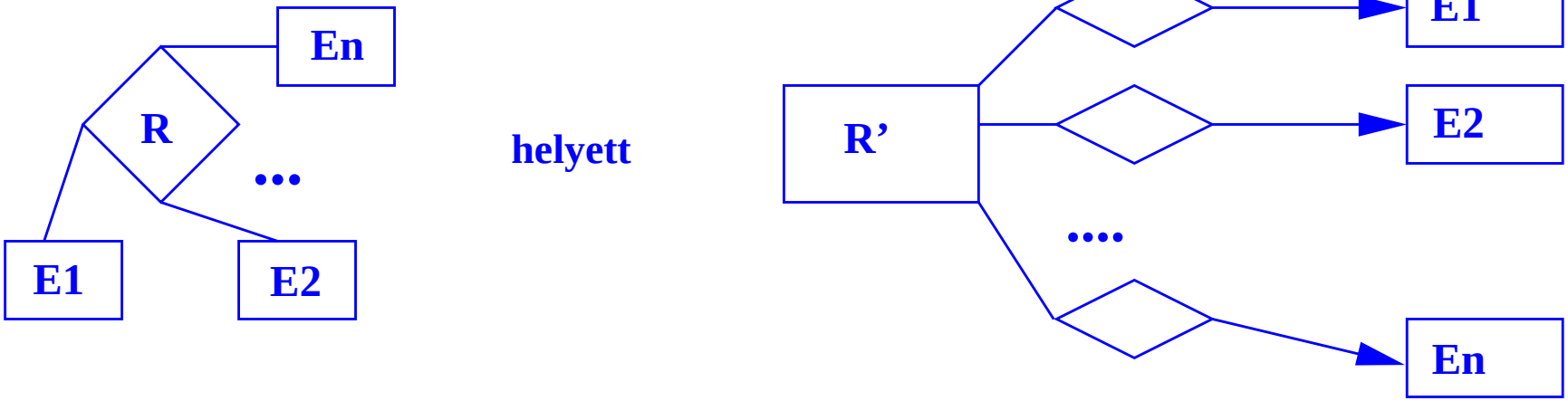
Sokágú kapcsolat átírása binárisra

Miért kell/jó ez?

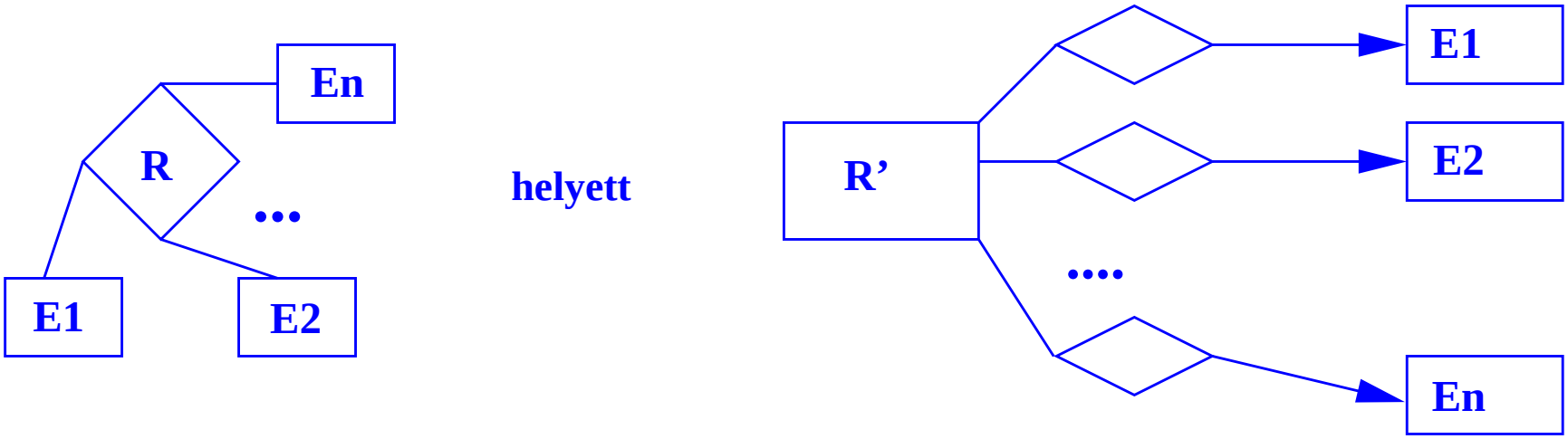
- átláthatóbb a bináris
- a megvalósításban könnyebben kezelhető
- látszik, hogy nem baj, hogy az ODL csak binárist tud, hisz a többágút is át lehet ilyenné írni

Az átíráshoz fő ötlet: egy $R(E_1, E_2, \ldots, E_n)$ kapcsolat egy eleme egy n egyedből álló vektor: $(e_1, e_2, \ldots, e_n)$.
Az átírásnál létrehozunk egy új egyedhalmazt, melynek az ilyen vektorok lesznek az egyedei és az ilyen vektorokat bináris több-egy kapcsolatok (n darab) fogják az $E_1, \ldots, E_n$ egyedhalmazokhoz kötni.

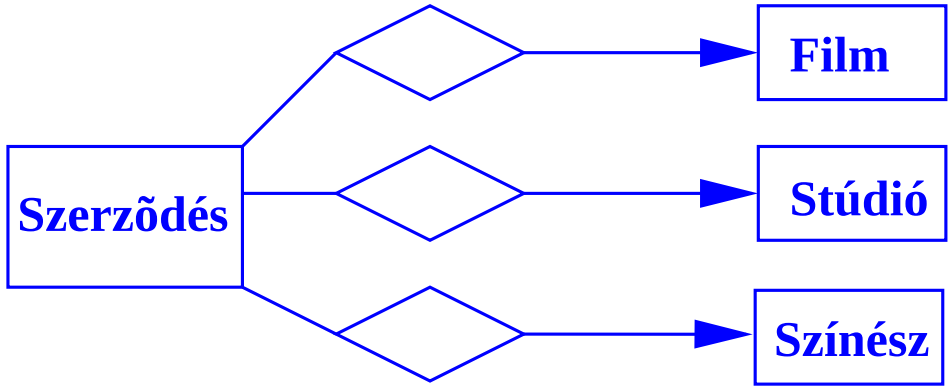
Rajzon:



Rajzon:



Konkrét példa (az előbbi háromágú szerződéses példa átírva):



Megjegyzések az átíráshoz

- Ez az átírás azért is jó, mert egy kapcsolatot majd úgyis valahogy így tudunk kezelni a fizikai megvalósításban: a kapcsolat egy eleme egy három mutatóból álló tömb lesz, ahol ez egyes mutatók a kapcsolatot alkotó egyedekre (film, színész, stúdió) mutatnak.

Megjegyzések az átíráshoz

- Ez az átírás azért is jó, mert egy kapcsolatot majd úgyis valahogy így tudunk kezelni a fizikai megvalósításban: a kapcsolat egy eleme egy három mutatóból álló tömb lesz, ahol ez egyes mutatók a kapcsolatot alkotó egyedekre (film, színész, stúdió) mutatnak.
- Az átírásnál persze veszíthetünk infót: az előbbi példánál elveszett az, hogy a többes kapcsolat „egy” volt a Stúdió fele. Ez nem baj, ezt majd a végén a relációs modellben úgyis finomabban le tudjuk írni.

Megjegyzések az átíráshoz

- Ez az átírás azért is jó, mert egy kapcsolatot majd úgyis valahogy így tudunk kezelni a fizikai megvalósításban: a kapcsolat egy eleme egy három mutatóból álló tömb lesz, ahol ez egyes mutatók a kapcsolatot alkotó egyedekre (film, színész, stúdió) mutatnak.
- Az átírásnál persze veszíthetünk infót: az előbbi példánál elveszett az, hogy a többes kapcsolat „egy” volt a Stúdió fele. Ez nem baj, ezt majd a végén a relációs modellben úgyis finomabban le tudjuk írni.
- Mivel minden többágú kapcsolatot át lehet írni binárissá, azért elég volna csak ilyen bináris kapcsolatokat használni.

Megjegyzések az átíráshoz

- Ez az átírás azért is jó, mert egy kapcsolatot majd úgyis valahogy így tudunk kezelni a fizikai megvalósításban: a kapcsolat egy eleme egy három mutatóból álló tömb lesz, ahol ez egyes mutatók a kapcsolatot alkotó egyedekre (film, színész, stúdió) mutatnak.
- Az átírásnál persze veszíthetünk infót: az előbbi példánál elveszett az, hogy a többes kapcsolat „egy” volt a Stúdió fele. Ez nem baj, ezt majd a végén a relációs modellben úgyis finomabban le tudjuk írni.
- Mivel minden többágú kapcsolatot át lehet írni binárissá, azért elég volna csak ilyen bináris kapcsolatokat használni.

Megjegyzések az átíráshoz

- Amúgy az ODL-ben lényegében ez történik, ha ott akarnánk ábrázolni azt a hármas kapcsolatot, ami a filmeket, színészeket és stúdiókat összeköti, akkor ehhez fel kellene vennünk egy negyedik osztályt így (és ez lényegében ugyanaz a helyzet, amit az E/K modellben kaptunk az átíráskor):

```
interface Szerződés {
        relationship Stúdió gyártja;
            inverse Stúdió::StúdióSzerződése;
        relationship Film filmje;
            inverse Film::FilmSzerződése;
        relationship Színész szereplője;
            inverse Színész::SzínészSzerződése;
                };
```

Feladat

Javasoljon ODL-sémát egy olyan banki adatbázishoz, amely tartalmazza az ügyfeleket és azok számláit.

Feladat

Javasoljon ODL-sémát egy olyan banki adatbázishoz, amely tartalmazza az ügyfeleket és azok számláit.
Az ügyfelekről tartsuk nyilván a nevüket, címüket, telefonszámukat és személyi számukat. A számláknak legyen számlaszámuk, típusuk (takarékbetét számla, folyószámla pl.) és egyenlegük.

Feladat

Javasoljon ODL-sémát egy olyan banki adatbázishoz, amely tartalmazza az ügyfeleket és azok számláit.
Az ügyfelekről tartsuk nyilván a nevüket, címüket, telefonszámukat és személyi számukat. A számláknak legyen számlaszámuk, típusuk (takarékbetét számla, folyószámla pl.) és egyenlegük.

Megoldás (néhány megoldás a sok lehetséges közül):

```
interface Ügyfél {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```

Megoldás (néhány megoldás a sok lehetséges közül):

```
interface Ügyfél {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};

interface Számla {
    attribute int számlaszám;
    attribute string típus;
    attribute int egyenleg;
    relationship Set<Ügyfél> tulajdonosai;
        inverse Ügyfél::számlái;
};
```

Variációk

- Lehetett volna struktúra a lakcím típusa:

```
interface Ügyfél {
    attribute string név;
    attribute Struct Cím{string város, string utca, int házszám} lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```

Variációk

- Lehetett volna struktúra a lakcím típusa:

```
interface Ügyfél {
    attribute string név;
    attribute Struct Cím{string város, string utca, int házszám} lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```

- Ha több címe is lehet egy embernek:

```
interface Ügyfél {
    attribute string név;
    attribute Set< Struct Cím{string város, string utca, int házszám}> lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```

Variációk

- Lehetett volna felsorolástípus a számla típusa:

```
interface Számla {
    attribute int számlaszám;
    attribute enum Típus{ betét, folyó } számlaTípus;
    attribute int egyenleg;
    relationship Set<Ügyfél> tulajdonosai;
        inverse Ügyfél::számlái;
};
```

Variációk

- Ha egy embernek csak egy számlája lehet és egy számlának csak egy tulajdonosa:

```
interface Ügyfél {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Számla számlája;
        inverse Számla::tulajdonosa;
};
```

Variációk

- Ha egy embernek csak egy számlája lehet és egy számlának csak egy tulajdonosa:

```
interface Ügyfél {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Számla számlája;
        inverse Számla::tulajdonosa;
};

interface Számla {
    attribute int számlaszám;
    attribute string típus;
    attribute int egyenleg;
    relationship Ügyfél tulajdonosa;
        inverse Ügyfél::számlája;
};
```

Variációk

- Ha egy embernek több lakhelye lehet és az egyes lakhelyekhez lehet több telefonszám is:

Variációk

- Ha egy embernek több lakhelye lehet és az egyes lakhelyekhez lehet több telefonszám is:
  attribute Set< Struct Cím{ string lakcím, Set<int> telszámok }> lakcím;

Variációk

- Ha egy embernek több lakhelye lehet és az egyes lakhelyekhez lehet több telefonszám is:
  attribute Set< Struct Cím{ string lakcím, Set<int> telszámok }> lakcím;
  Struct-on belül nem lehet kollekciótípus illetve két kollekciótípus sem lehet egymásba ágyazva egy attribútum típusában.

Variációk

- Ha egy embernek több lakhelye lehet és az egyes lakhelyekhez lehet több telefonszám is:
  attribute Set< Struct Cím{ string lakcím, Set<int> telszámok }> lakcím;
  Struct-on belül nem lehet kollekciótípus illetve két kollekciótípus sem lehet egymásba ágyazva egy attribútum típusában.
  Az se lenne jó, ha külön nyilvántartunk egy csomó lakcímet és ettől függetlenül az összes telefonszámot, mert akkor nem fogjuk tudni, hogy melyik címhez melyik szám tartozik.

```
interface Ügyfél {
    attribute string név;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla: tulajdonosai;
    relationship Set<Lakhely> ittLakik;
        inverse Lakhely::lakói;
};
interface Számla {
    attribute int számlaszám;
    attribute string típus;
    attribute int egyenleg;
    relationship Set<Ügyfél> tulajdonosai;
        inverse Ügyfél::számlái;
};
```

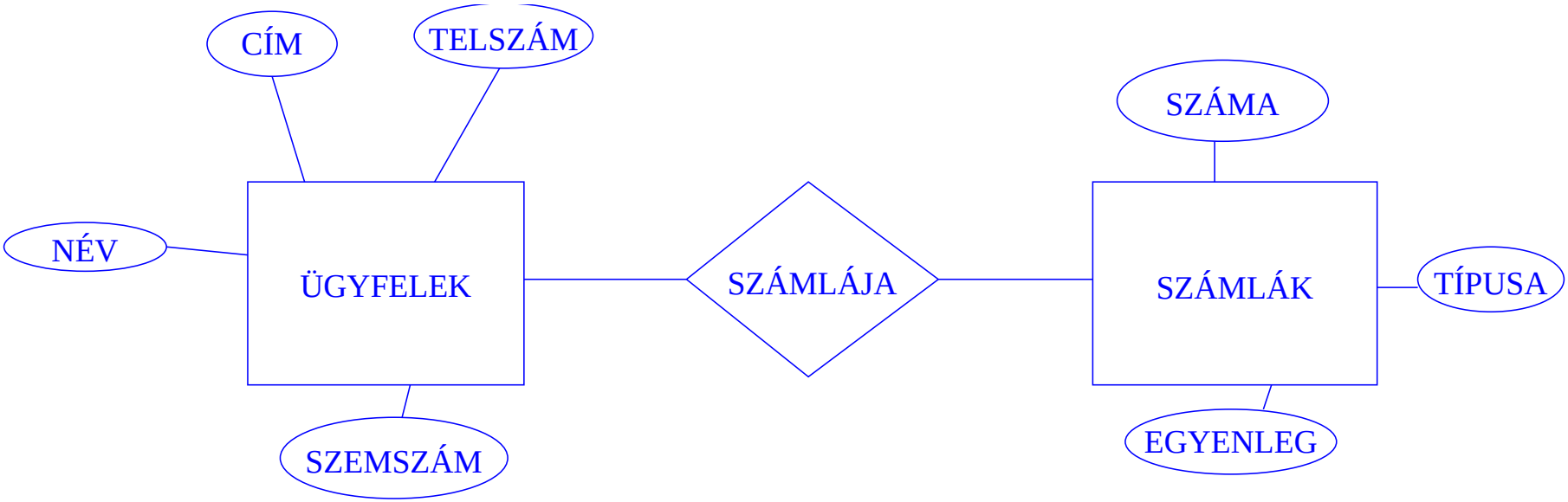
```
interface Lakhely {
    attribute string város;
    attribute string utca;
    attribute int házszám;
    attribute Set<int> telefonszámok;
    relationship Set<Ügyfél> lakói;
        inverse Ügyfél::ittLakik;
};
```

Feladat

Tervezzen E/K diagrammot egy olyan banki adatbázishoz, amely tartalmazza az ügyfeleket és azok számláit.
Az ügyfelekről tartsuk nyilván a nevüket, címüket, telefonszámukat és személyi számukat. A számláknak legyen számlaszámuk, típusuk (takarékbetét számla, folyószámla pl.) és egyenlegük.

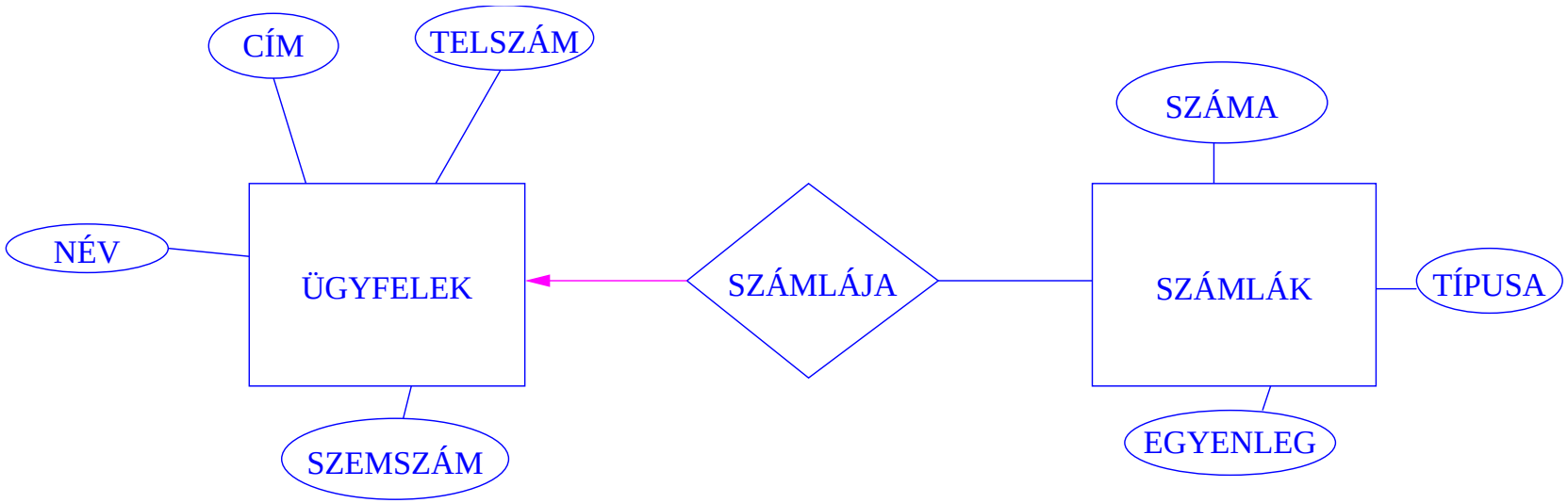
Feladat

Tervezzen E/K diagrammot egy olyan banki adatbázishoz, amely tartalmazza az ügyfeleket és azok számláit.
Az ügyfelekről tartsuk nyilván a nevüket, címüket, telefonszámukat és személyi számukat. A számláknak legyen számlaszámuk, típusuk (takarékbetét számla, folyószámla pl.) és egyenlegük.
Megoldás (néhány megoldás a sok lehetséges közül):



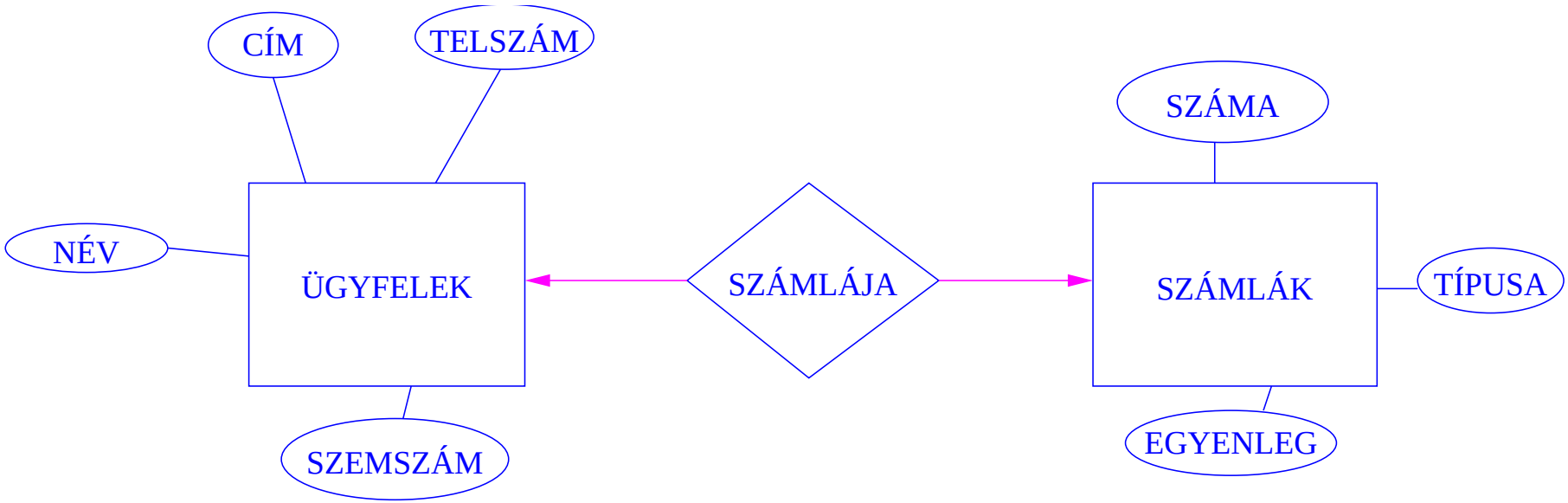
Variációk

- Ha egy számlának csak egy tulajdonosa lehet:



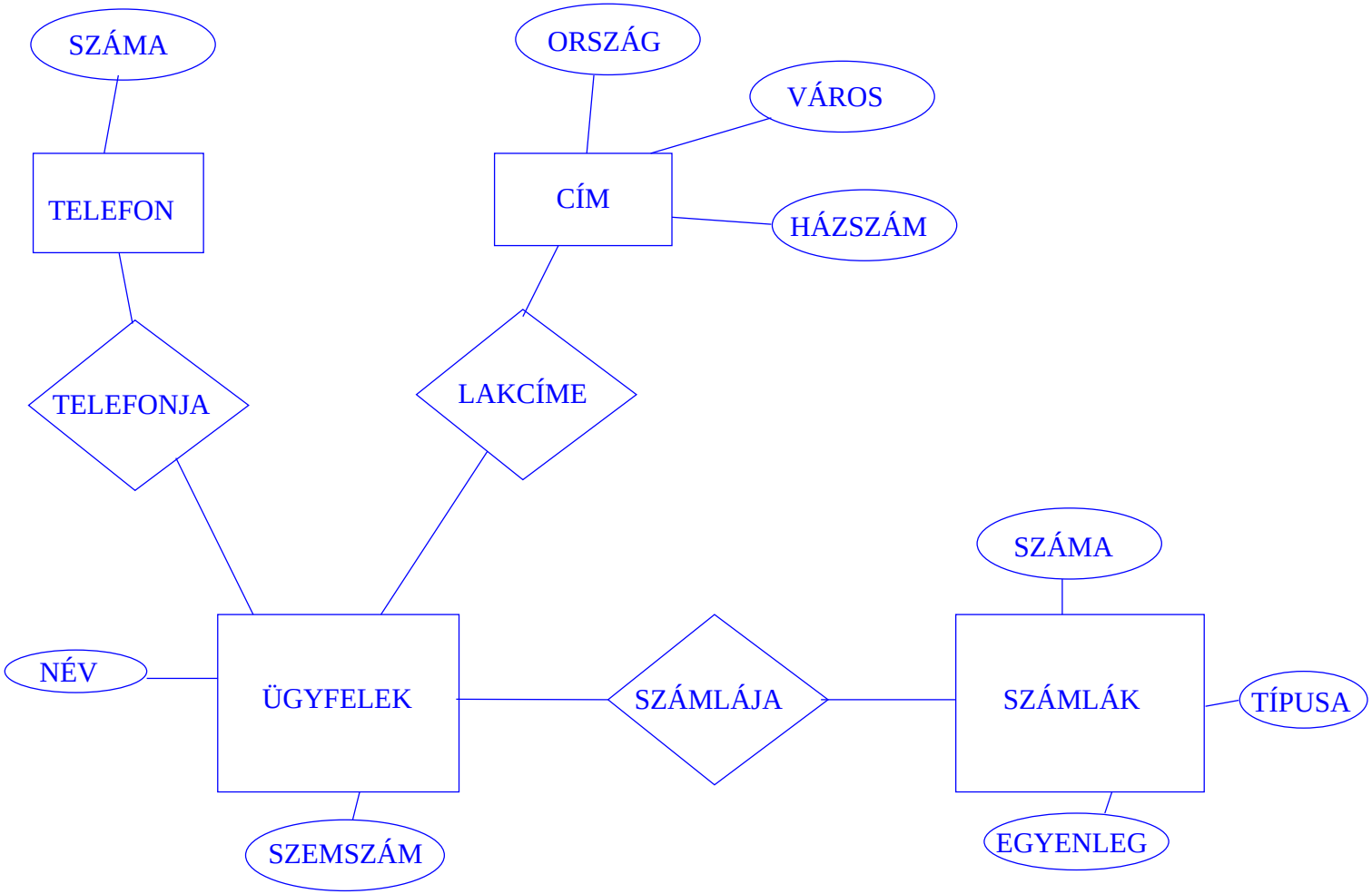
Variációk

- Ha egy számlának csak egy tulajdonosa lehet és egy ügyfélnek csak egy számlája lehet:



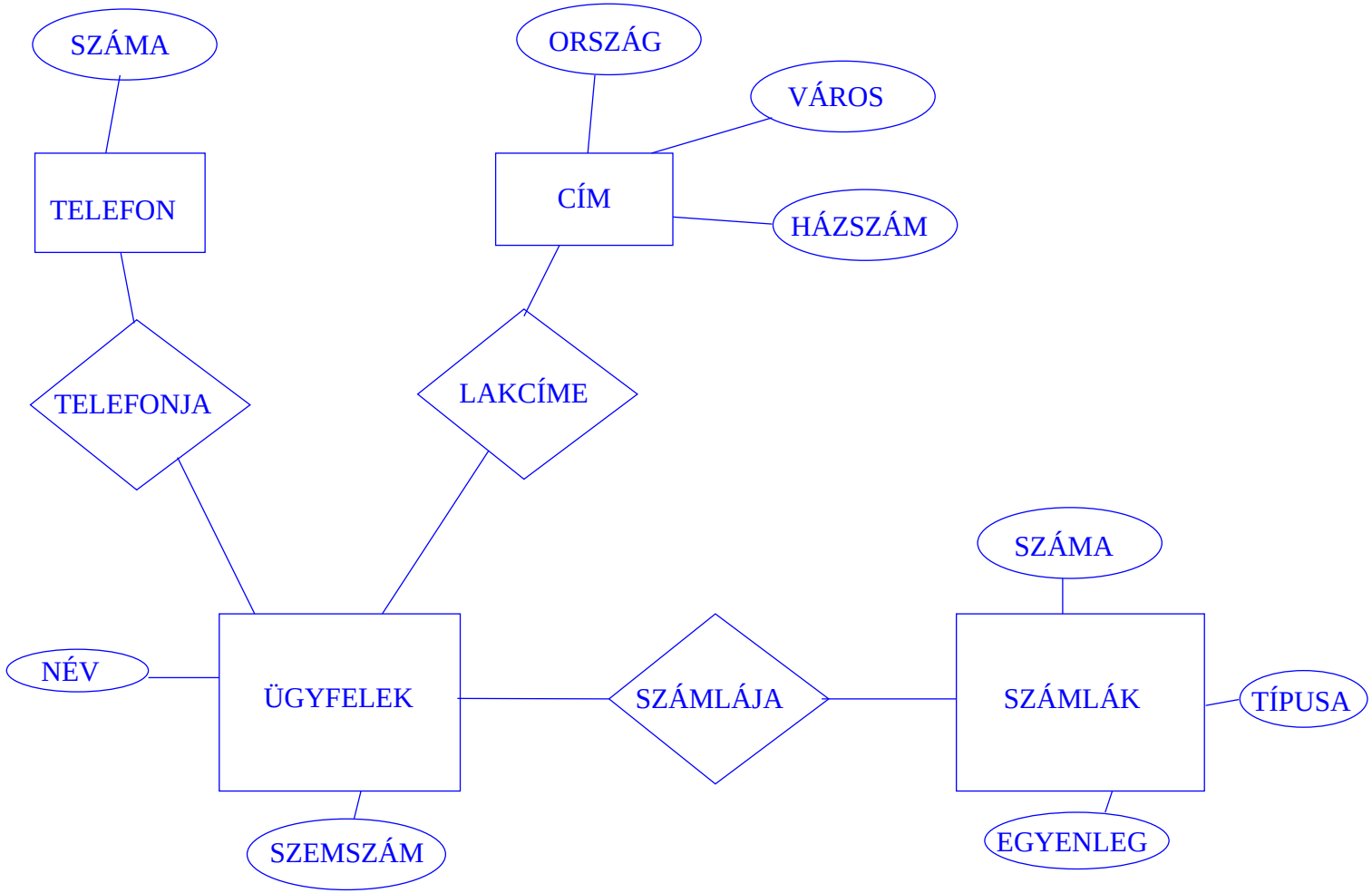
Variációk

- Ha egy embernek több címe és több telefonszáma is lehet, de azt nem kell nyilvántartani, hogy mik az összetartozó lakcím-telefonszám párok:



Variációk

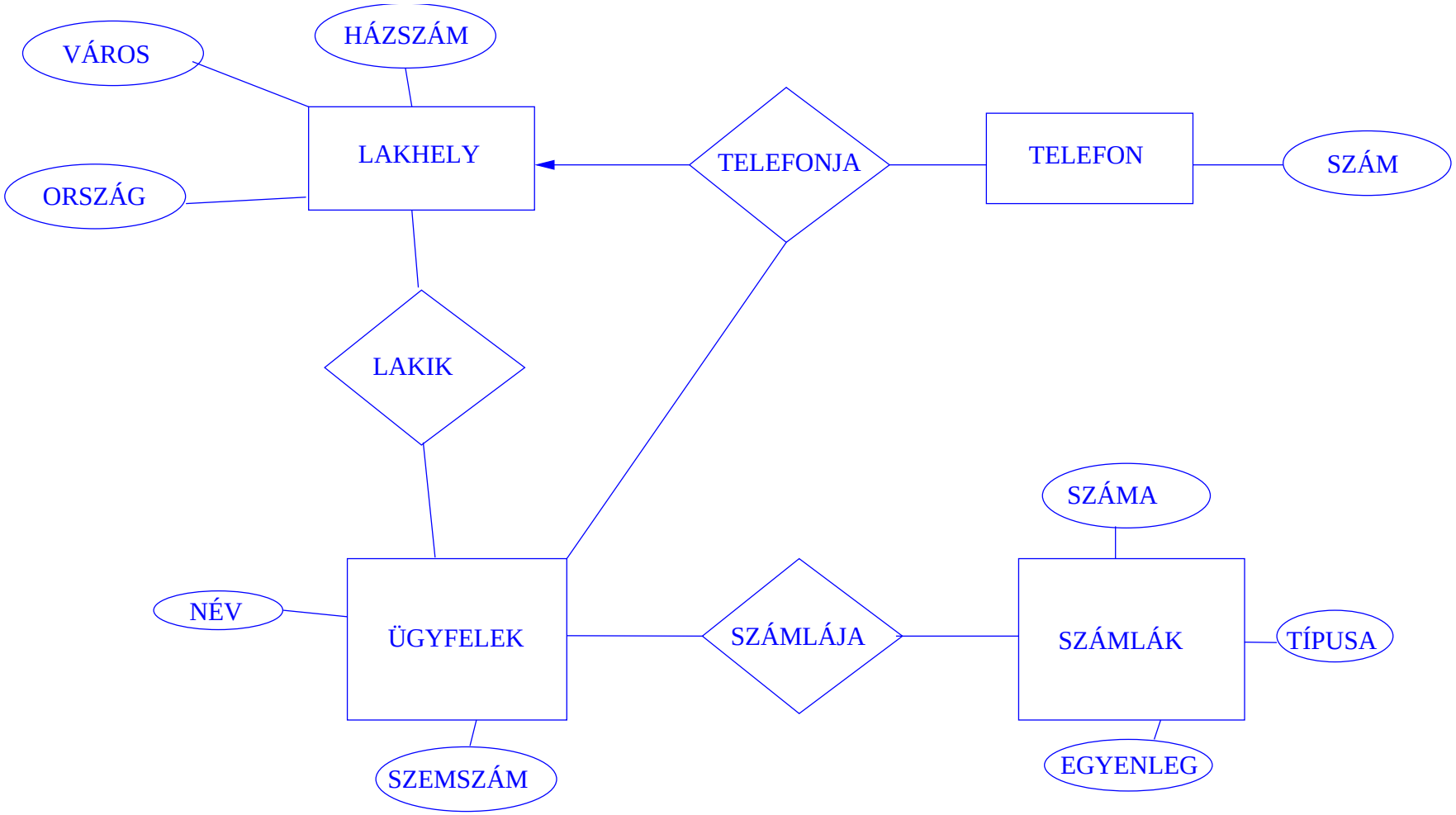
- Ha egy embernek több címe és több telefonszáma is lehet, de azt nem kell nyilvántartani, hogy mik az összetartozó lakcím-telefonszám párok:



Azért nem lehetett az „ügyfelek” egyedhalmaz attribútuma a telefonszám, mert csak egyszerű típusokat használhatunk, kollekciókat nem.

Variációk

- Ha azt is nyilván kell tartani, hogy mik az összetartozó lakcím- telefonszám párok:



Alosztályok

Egy osztály (egyedhalmaz) speciális tulajdonságú, de egymáshoz hasonló objektumai (egyedei) alkotják. Ezeket érdemes együtt kezelni, de úgy, hogy a (nagyobb) osztályba való tartozásuk is megmaradjon.

Alosztályok

Egy osztály (egyedhalmaz) speciális tulajdonságú, de egymáshoz hasonló objektumai (egyedei) alkotják. Ezeket érdemes együtt kezelni, de úgy, hogy a (nagyobb) osztályba való tartozásuk is megmaradjon.

ODL-ben

Pl. a Film a osztályon belül akarhatunk külön Rajzfilm, Vígjáték, Krimifilm alosztályokat. Ennek megadása az osztályok deklarációjakor:
A Film osztályt megadjuk, úgy, mint eddig, és

```
interface Rajzfilm:Film{
        relationship Set<Színész> hangok;
            inverse Színész::hangItt;
                    };
```

Vagyis először megadjuk az alosztály nevét, aztán : után annak az osztálynak a nevét, aminek ő alosztálya lesz. A zárójelen belül már csak azokat az attribútumokat/kapcsolatokat kell megadni, amik az alosztály sajátjai, a többi attribútumot/kapcsolatot örökli a főosztálytól.

Példa még:

```
interface Krimifilm:Film{
        attribute Set⟨string⟩ bizonyítékok;
                };
```

Példa még:

```
interface Krimifilm:Film{
        attribute Set⟨string⟩ bizonyítékok;
                };
```

Sőt, lehet több szintű öröklés, illetve egy alosztály örökölhet két főosztálytól is:

```
interface Krimirajzfilm:Krimifilm, Rajzfilm{
                };
```

Itt a Krimirajzfilm alosztály örökli a Krimifilm és a Rajzfilm (al)osztály összes attribútumát és kapcsolatát (természetesen azokat is, amiket azok is úgy örököltek), neki magának pedig semmi saját attribútuma/kapcsolata nincsen.

Példa még:

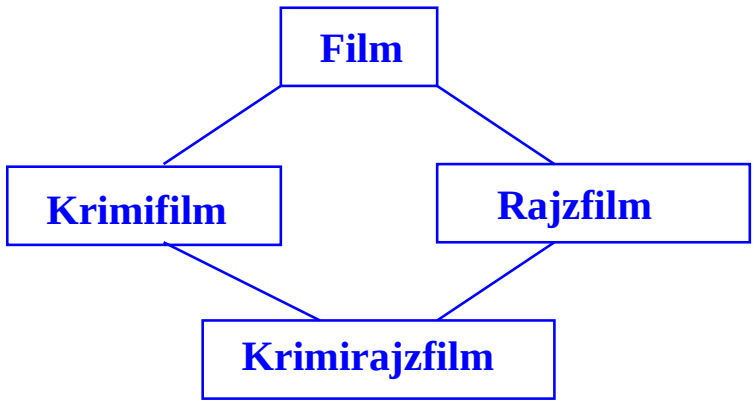
```
interface Krimifilm:Film{
        attribute Set⟨string⟩ bizonyítékok;
                };
```

Sőt, lehet több szintű öröklés, illetve egy alosztály örökölhet két főosztálytól is:

```
interface Krimirajzfilm:Krimifilm, Rajzfilm{
                };
```

Itt a Krimirajzfilm alosztály örökli a Krimifilm és a Rajzfilm (al)osztály összes attribútumát és kapcsolatát (természetesen azokat is, amiket azok is úgy örököltek), neki magának pedig semmi saját attribútuma/kapcsolata nincsen.

Az öröklési kép ilyen:



Ha olyan filmeket is nyilván akarunk tartani, amik rajzfilmek (akarunk hangok kapcsolatot a Színészek felé) és krimik is (akarunk bizonyítékok attribútumot), akkor muszáj létrehoznunk a fenti Krimirajzfilm osztályt, mert az objektumos szemléletben minden objektum csak egy (al)osztályhoz tartozhat. Ha nem lenne Krimirajzfilm osztály, akkor vagy csak a hangokat vagy csak a bizonyítékokat tudnánk feljegyezni.

Ha olyan filmeket is nyilván akarunk tartani, amik rajzfilmek (akarunk hangok kapcsolatot a Színészek felé) és krimik is (akarunk bizonyítékok attribútumot), akkor muszáj létrehoznunk a fenti Krimirajzfilm osztályt, mert az objektumos szemléletben minden objektum csak egy (al)osztályhoz tartozhat. Ha nem lenne Krimirajzfilm osztály, akkor vagy csak a hangokat vagy csak a bizonyítékokat tudnánk feljegyezni. Probléma lehet a többszörös öröklődésnél, ha ugyanolyan nevű attribútumot/kapcsolatot több helyről is örökölne egy alosztály. Ilyenkor valamelyiket át kell nevezni.

Alosztályok az E/K modellben

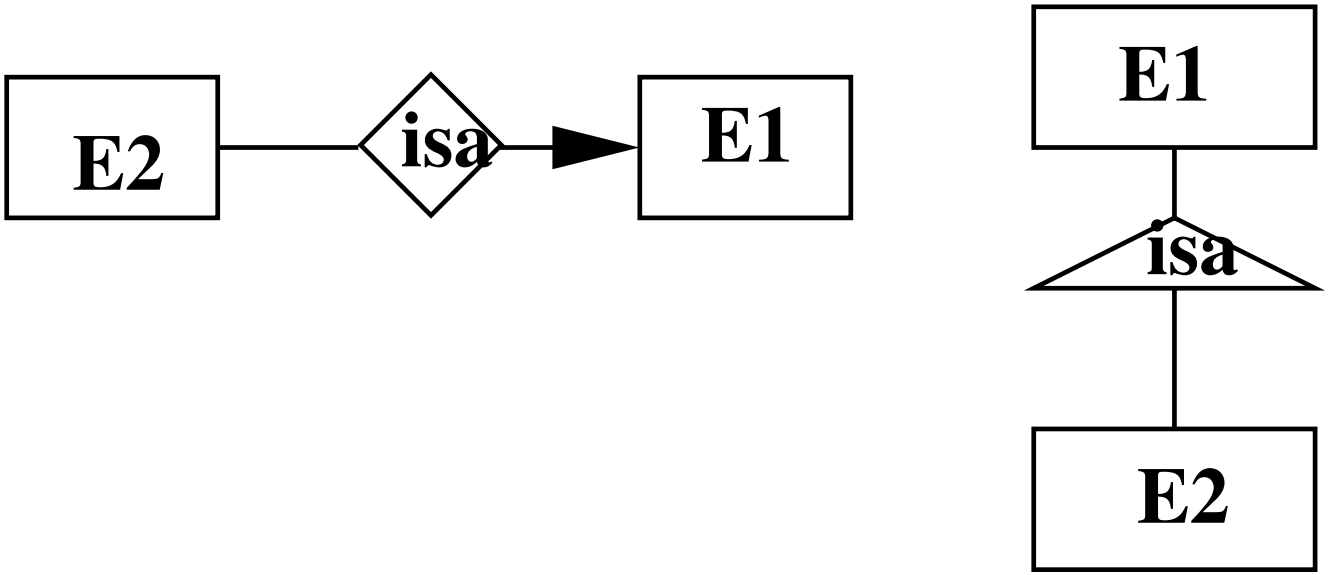
Sokkal egyszerűbb, mint az ODL-ben, de a cél ugyanaz: egy egyedhalmaz speciális egyedeit együtt kezelni. Ehhez egy speciális (*isa*, magyarul *azegy*) kapcsolat van.

Alosztályok az E/K modellben

Sokkal egyszerűbb, mint az ODL-ben, de a cél ugyanaz: egy egyedhalmaz speciális egyedeit együtt kezelni. Ehhez egy speciális (*isa*, magyarul *azegy*) kapcsolat van.
(Motiváció: A ló az egy állat = a lovak alosztályát alkotják az állatoknak.)

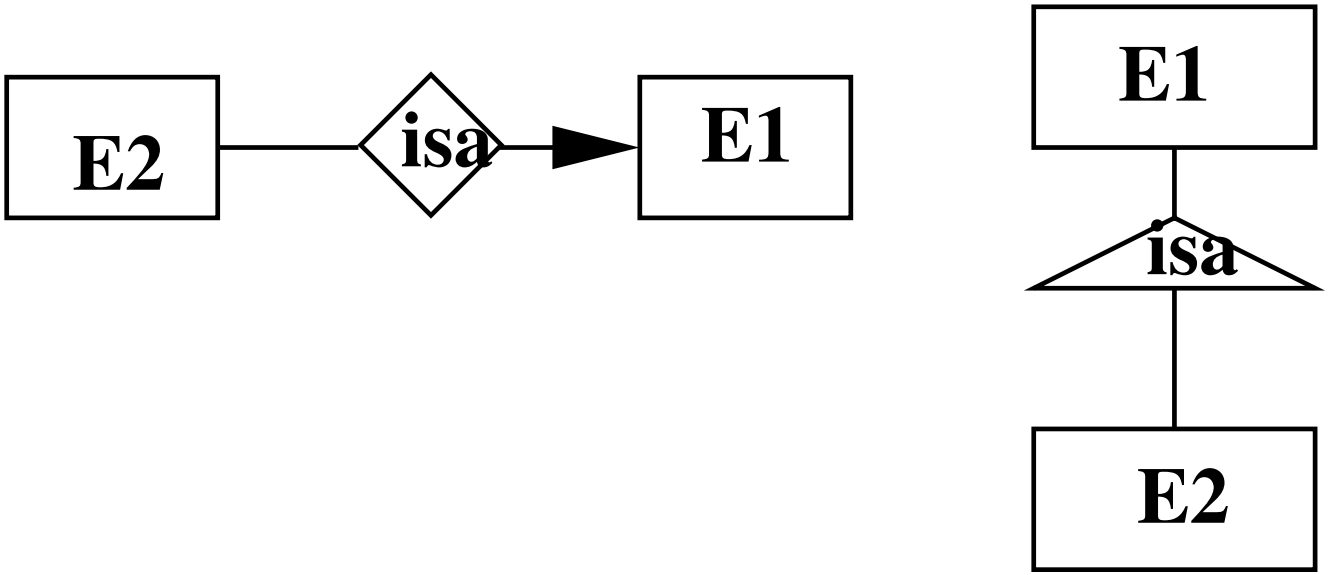
Alosztályok az E/K modellben

Sokkal egyszerűbb, mint az ODL-ben, de a cél ugyanaz: egy egyedhalmaz speciális egyedeit együtt kezelni. Ehhez egy speciális (*isa*, magyarul *azegy*) kapcsolat van.
(Motiváció: A ló az egy állat = a lovak alosztályát alkotják az állatoknak.)
Jelölés, ha E_2 alosztálya E_1-nek:



Alosztályok az E/K modellben

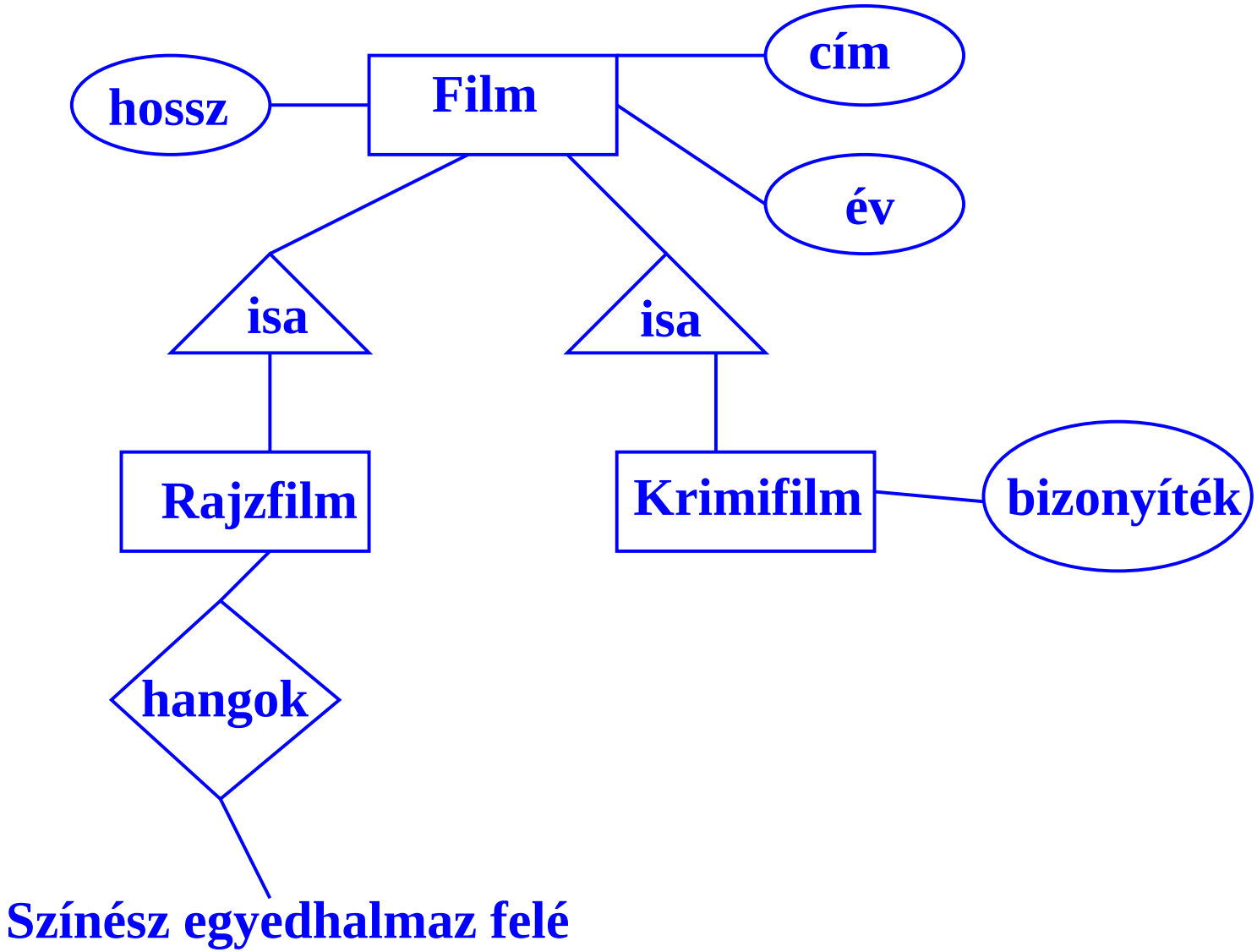
Sokkal egyszerűbb, mint az ODL-ben, de a cél ugyanaz: egy egyedhalmaz speciális egyedeit együtt kezelni. Ehhez egy speciális (*isa*, magyarul *azegy*) kapcsolat van.
(Motiváció: A ló az egy állat = a lovak alosztályát alkotják az állatoknak.)
Jelölés, ha E_2 alosztálya E_1-nek:



Az első esetben a nyíl arra mutat, amelyik a Fő osztály, ez most nem a több-egy kapcsolatnál megszokott nyíl, az *isa* ilyen szempontból is speciális kapcsolat. A második esetben az alosztály van alul. Az alárendelt halmaz most is örökli a főosztály attribútumait és kapcsolatait, de persze lehetnek neki sajátjai is.

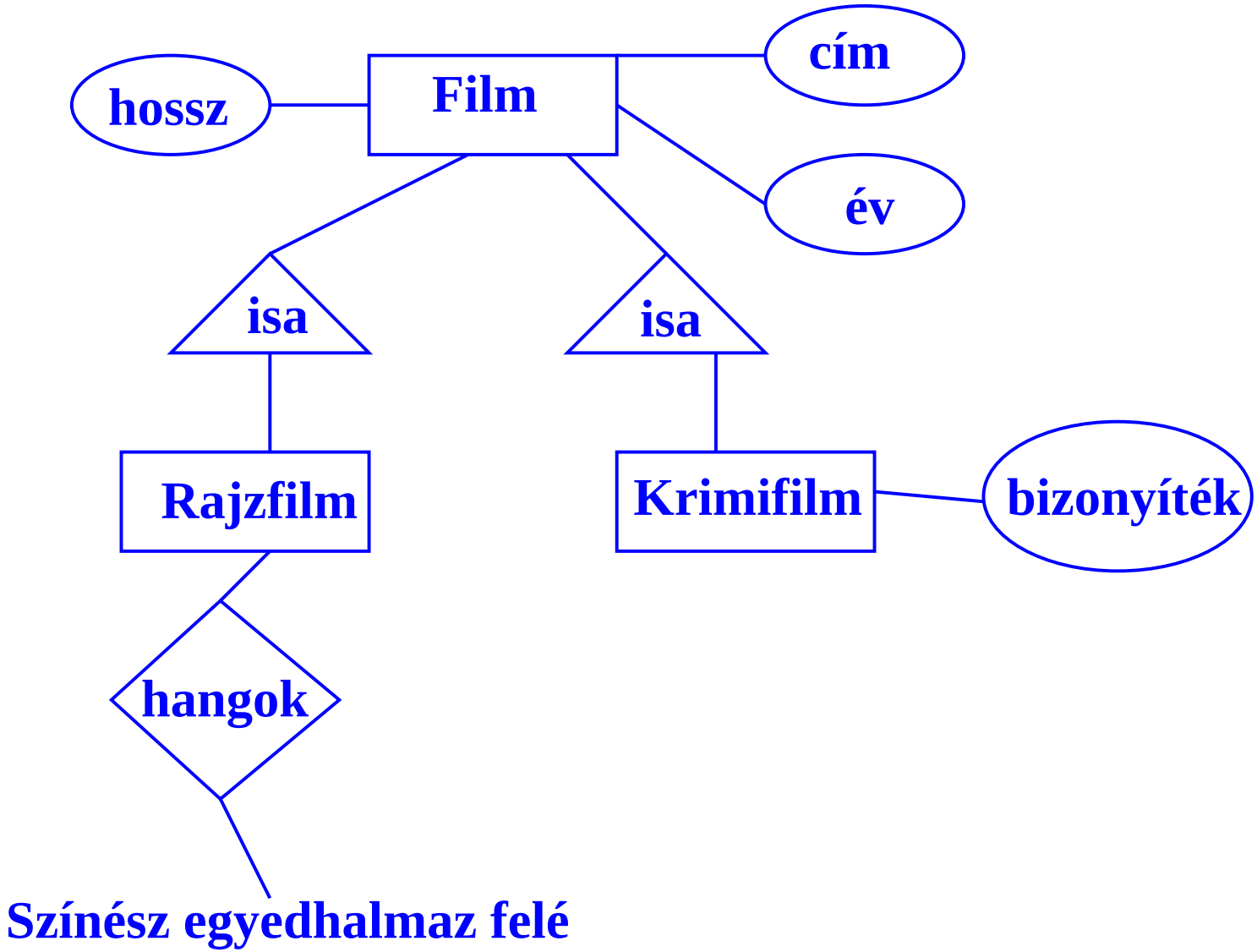
Különbségek a két modell között

A filmes példánál ez lesz az E/K modellben:



Különbségek a két modell között

A filmes példánál ez lesz az E/K modellben:



Különbségek a két modell között

Az ODL-ben minden objektum csak egy osztályhoz tartozhat (ezért ott kellett Krimirajzfilm osztály), az E/K modellnél viszont lehetséges az, hogy egy egyed egyszerre több osztály/alosztály része is, ilyenkor az attribútumait/kapcsolatait úgy szedi össze a felmenőitől (E/K-ban nem kell Krimirajzfilm alosztály). A relációs modellre való átíráskor majd úgyis egységesen fogjuk ezeket a jellemzőket kezelni (lásd majd ott).

Különbségek a két modell között

Az ODL-ben minden objektum csak egy osztályhoz tartozhat (ezért ott kellett Krimirajzfilm osztály), az E/K modellnél viszont lehetséges az, hogy egy egyed egyszerre több osztály/alosztály része is, ilyenkor az attribútumait/kapcsolatait úgy szedi össze a felmenőitől (E/K-ban nem kell Krimirajzfilm alosztály). A relációs modellre való átíráskor majd úgyis egységesen fogjuk ezeket a jellemzőket kezelni (lásd majd ott).
Így az E/K modellben egy olyan film, ami krimi is és rajzfilm is (pl. Macskafogó), három helyről szedi össze az attribútumait/kapcsolatait: a címét, hosszát és gyártási évét a Film egyedhalmazból, a hangjait a Rajzfilm alosztályból, a bizonyítékot pedig a Krimifilmből.

Példa

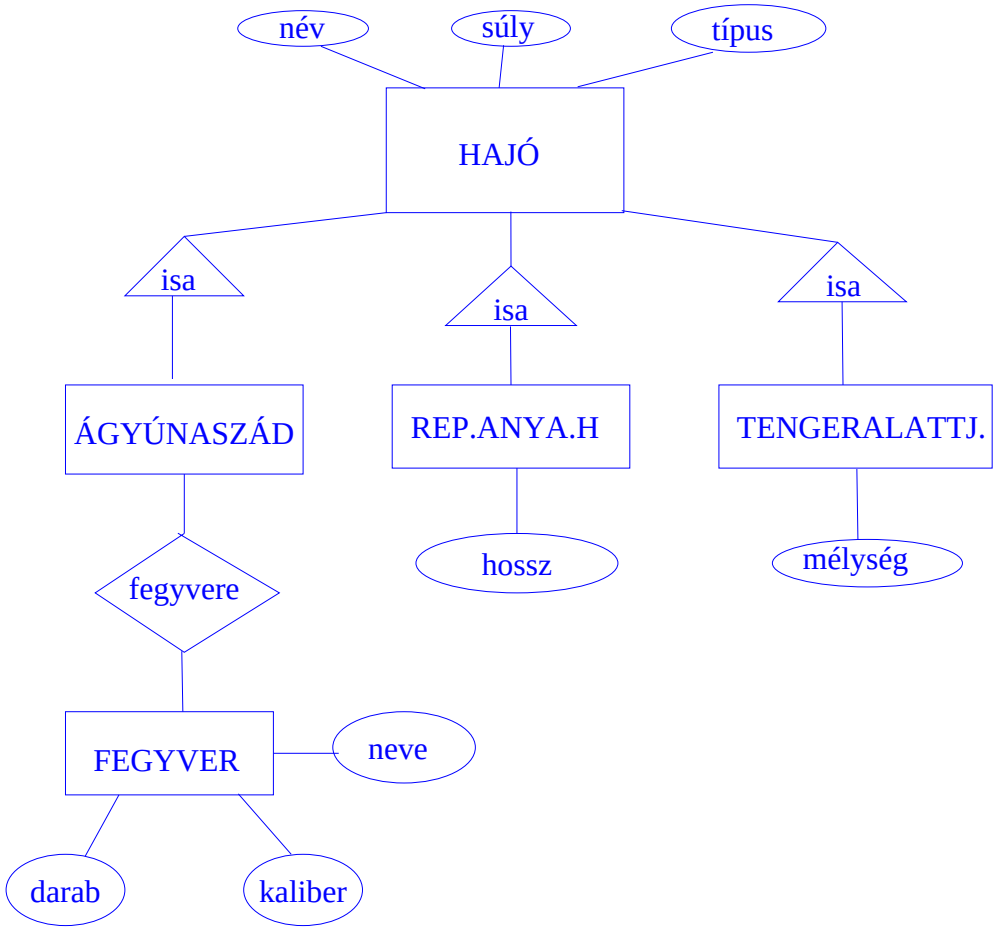
Hadihajók adatbázisát szeretnénk megadni ODL-ben és E/K diagrammal. Minden hadihajóról nyilvántartjuk a nevét, a vízkiszorítását tonnában, típusát. Négyfajta hajót akarunk nyilvántartani:

1. Ágyúnaszád (itt nyilvántartjuk a fegyverek számát és kaliberét)
2. Repülőgép-anyahajó (tároljuk a leszállópálya hosszát)
3. Tengeralattjáró (max. merülési mélység)
4. Csata-repülőgép-anyahajó (olyan ágyúnaszád, ami repülőgépanyahajó is).

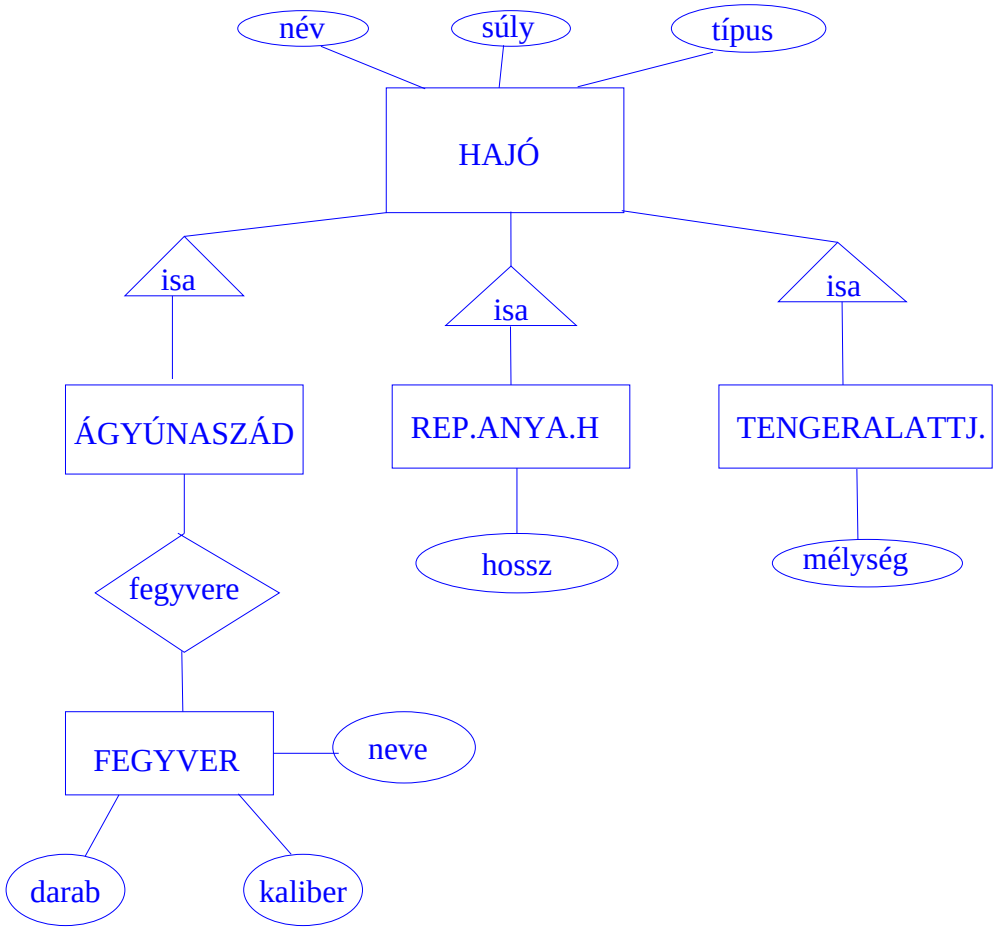
Megoldás ODL-ben

```
interface Hajó {
    attribute string név;
    attribute int súly;
    attribute string típus;
};

interface Ágyúnaszád:Hajó {
    attribute Set<Struct Fegyver{string név, int darab, int kaliber}> fegyverek;
};

interface Repülőgép-anyahajó:Hajó {
    attribute int hossz;
};

interface Tengeralattjáró:Hajó {
    attribute int mélység;
};

interface Csatarepülőgép-anyahajó:Ágyúnaszád, Repülőgép-anyahajó { };
```

Megoldás E/K modellel



Megoldás E/K modellel



Megjegyzés: Itt nem kell külön egyedhalmaz a Csatarepülőgép-anyahajóknak, mert nincs olyan attribútum, ami csak ezeknél lenne. Egy ilyen hajót úgy tartunk majd nyilván, hogy lesznek attribútumai mind az ágyúnaszádoktól, mind a repülőgép-anyahajóktól.

Megszorítások

Olyan megszorításokat is ábrázolni akarunk az adathalmazokon, attribútumokon, kapcsolatokon, amik nem fejezhetők ki pusztán az attribútumok és a kapcsolatok felsorolásával.
Ezek további infók, amik

- a séma részei, ezért már a tervezéskor kell rájuk gondolni,

Megszorítások

Olyan megszorításokat is ábrázolni akarunk az adathalmazokon, attribútumokon, kapcsolatokon, amik nem fejezhetők ki pusztán az attribútumok és a kapcsolatok felsorolásával.
Ezek további infók, amik

- a séma részei, ezért már a tervezéskor kell rájuk gondolni,
- olyan megkötéseket tartalmaznak, amikre majd mindig figyelni kell.

Megszorítások

Olyan megszorításokat is ábrázolni akarunk az adathalmazokon, attribútumokon, kapcsolatokon, amik nem fejezhetők ki pusztán az attribútumok és a kapcsolatok felsorolásával.

Ezek további infók, amik

- a séma részei, ezért már a tervezéskor kell rájuk gondolni,
- olyan megkötéseket tartalmaznak, amikre majd mindig figyelni kell.

Nem világos, hogy mik a jó megkötések, miket lehet jól kezelni.

Megszorítások

Olyan megszorításokat is ábrázolni akarunk az adathalmazokon, attribútumokon, kapcsolatokon, amik nem fejezhetők ki pusztán az attribútumok és a kapcsolatok felsorolásával.

Ezek további infók, amik

- a séma részei, ezért már a tervezéskor kell rájuk gondolni,
- olyan megkötéseket tartalmaznak, amikre majd mindig figyelni kell.

Nem világos, hogy mik a jó megkötések, miket lehet jól kezelni.

Megszorítások

Típusai (tipikus, (néha) jól kezelhető megszorítások):

- *Kulcsok*: olyan attribútum, vagy attribútumhalmaz megadása, ami az egyedet/objektumot már egyértelműen azonosítja. (Pl. személyi szám vagy filmnél gyártási év és cím.) A tervezéskor döntjük el, hogy mik alkossanak kulcsot (persze a valóságot szem előtt tartva). A kulcshoz tartozó attribútumoknak értékeket adva, legfeljebb egy objektum vagy egyed létezhet, amikhez ezek az értékek tartoznak.

Megszorítások

Típusai (tipikus, (néha) jól kezelhető megszorítások):

- *Kulcsok*: olyan attribútum, vagy attribútumhalmaz megadása, ami az egyedet/objektumot már egyértelműen azonosítja. (Pl. személyi szám vagy filmnél gyártási év és cím.) A tervezéskor döntjük el, hogy mik alkossanak kulcsot (persze a valóságot szem előtt tartva). A kulcshoz tartozó attribútumoknak értékeket adva, legfeljebb egy objektum vagy egyed létezhet, amikhez ezek az értékek tartoznak. Néha tűnhet úgy az aktuális adatokból, hogy valami kulcs (mert akkor éppen nincs két egyed ugyanolyan értékekkel), de ettől még nem lesz kulcs valami, az csak a deklarációtól függ.
- *Egyértékűségi megszorítások*: előírhatjuk, hogy valami érték vagy értékkombináció legyen egyedi. Pl. ilyen a kulcsok megadása, vagy az, hogy egy kapcsolat nem rendelhet halmazt értékül egy egyedhez/objektumhoz. Ilyenek lesznek majd a funkcionális függések is a relációs modellben.

Megszorítások

- *Hivatkozási épség*: a hivatkozott dolognak léteznie kell.

Megszorítások

- *Hivatkozási épség*: a hivatkozott dolognak léteznie kell.
- *Értelmezési tartomány korlátozása*: attribútum lehetséges értékeire megkötés (pl. magasság legyen kisebb 300-nál, film gyártási éve 1800 utáni). *Módszerek erre:* típusok megadása, felsorolás típus, konkrétabban majd az SQL DDL-jénél.

Megszorítások

- *Hivatkozási épség*: a hivatkozott dolognak léteznie kell.
- *Értelmezési tartomány korlátozása*: attribútum lehetséges értékeire megkötés (pl. magasság legyen kisebb 300-nál, film gyártási éve 1800 utáni). *Módszerek erre:* típusok megadása, felsorolás típus, konkrétabban majd az SQL DDL-jénél.
- *Egyéb megszorítások*: minden más, pl. kapcsolat fokának korlátozása (egy filmnek max. 10 szereplőjét akarjuk nyilvántartani).

Általános gond: Mik a jó megszorítások? Miket lehet megvalósítani?

Általános gond: Mik a jó megszorítások? Miket lehet megvalósítani?
Ez persze majd a konkrét megvalósítástól függ, a konkrét DDL-től, de azért jó lenne már a modellezéskor is annyit leírni, amennyit csak lehet.

Általános gond: Mik a jó megszorítások? Miket lehet megvalósítani?
Ez persze majd a konkrét megvalósítástól függ, a konkrét DDL-től, de azért jó lenne már a modellezéskor is annyit leírni, amennyit csak lehet.

A megszorítások haszna

- jobban/valósághoz közelebbi módon le lehet velük írni a világot

Általános gond: Mik a jó megszorítások? Miket lehet megvalósítani?
Ez persze majd a konkrét megvalósítástól függ, a konkrét DDL-től, de azért jó lenne már a modellezéskor is annyit leírni, amennyit csak lehet.

A megszorítások haszna

- jobban/valósághoz közelebbi módon le lehet velük írni a világot
- segíthetik a tárolást (elég pl. a kulcsattribútumokat megadni kereséskor)

Megszorítások megadása ODL-ben

- *Kulcs*:
  - lehet egy vagy több kulcs

Megszorítások megadása ODL-ben

- *Kulcs*:
  - lehet egy vagy több kulcs
  - egy kulcs állhat egy vagy több attribútumból

Megszorítások megadása ODL-ben

- *Kulcs*:
  - lehet egy vagy több kulcs
  - egy kulcs állhat egy vagy több attribútumból

Megadása formailag: interface <Osztálynév> (Kulcsinfók){...}
ahol a Kulcsinfók = key(s) $K_1, \ldots, K_n$
ahol K_i egy kulcsleírás, ami
<attribútumnév>, ha a kulcs egy attribútumból áll vagy
$(< attr_1 >, \ldots, < attr_n >)$, ha a kulcs több attribútumos.

Megszorítások megadása ODL-ben

- *Kulcs*:
  - lehet egy vagy több kulcs
  - egy kulcs állhat egy vagy több attribútumból

Megadása formailag: interface <Osztálynév> (Kulcsinfók){. . . }
ahol a Kulcsinfók = key(s) $K_1, \ldots, K_n$
ahol K_i egy kulcsleírás, ami
<attribútumnév>, ha a kulcs egy attribútumból áll vagy
$(< attr_1 >, \ldots, < attr_n >)$, ha a kulcs több attribútumos.

Például:
interface Film (key (cím, év)) {. . . }
itt egy darab kulcs van, ami két attribútumból áll, ezek együtt azonosítanak egy objektumot

Megszorítások megadása ODL-ben

- *Kulcs*:
  - lehet egy vagy több kulcs
  - egy kulcs állhat egy vagy több attribútumból

Megadása formailag: interface <Osztálynév> (Kulcsinfók){...}
ahol a Kulcsinfók = key(s) $K_1, \ldots, K_n$
ahol K_i egy kulcsleírás, ami
<attribútumnév>, ha a kulcs egy attribútumból áll vagy
$(< attr_1 >, \ldots, < attr_n >)$, ha a kulcs több attribútumos.

Például:
interface Film (key (cím, év)) {...}
itt egy darab kulcs van, ami két attribútumból áll, ezek együtt azonosítanak egy objektumot

interface Dolgozó (key dolgozóID, tbszám) {...}
itt két egy-attribútumos kulcs van, mindegyik külön-külön azonosít

Korábbi példa

```
interface Ügyfél (key szemszám) {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```

Korábbi példa

```
interface Ügyfél (key szemszám) {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};

interface Számla (key számlaszám) {
    attribute int számlaszám;
    attribute string típus;
    attribute int egyenleg;
    relationship Set<Ügyfél> tulajdonosai;
        inverse Ügyfél::számlái;
};
```

Megszorítások megadása ODL-ben

- *Egyértékűség az ODL-ben*:
  - Kulcs-szerű megszorítás jól leírható (lásd előbb)

Megszorítások megadása ODL-ben

- *Egyértékűség az ODL-ben*:
  - Kulcs-szerű megszorítás jól leírható (lásd előbb)
  - Az attribútumok és a kapcsolatok többességének szabályozására: a kollekcióoperátorok használata/nem használata. Így előírható, hogy egy attribútum/kapcsolat csak egy értéket vehessen fel.

Megszorítások megadása ODL-ben

- *Egyértékűség az ODL-ben*:
  - Kulcs-szerű megszorítás jól leírható (lásd előbb)
  - Az attribútumok és a kapcsolatok többességének szabályozására: a kollekcióoperátorok használata/nem használata. Így előírható, hogy egy attribútum/kapcsolat csak egy értéket vehessen fel.
  - Egyértékűséget kétféleképpen is lehet érteni:
    - legfeljebb egy értéken vehessen fel valami (ekkor esetleg állhat NULL-érték is bizonyos helyeken, ami jelentheti azt, hogy nincs megfelelő érték, vagy hogy van, de nem ismert),
    - pontosan egyet vehessen fel (pl. kulcsattribútum nem lehet NULL).

    Hogy melyik megközelítés van, az rendszerfüggő.

Megszorítások megadása ODL-ben

- *Egyértékűség az ODL-ben*:
  - Kulcs-szerű megszorítás jól leírható (lásd előbb)
  - Az attribútumok és a kapcsolatok többességének szabályozására: a kollekcióoperátorok használata/nem használata. Így előírható, hogy egy attribútum/kapcsolat csak egy értéket vehessen fel.
  - Egyértékűséget kétféleképpen is lehet érteni:
    - legfeljebb egy értéken vehessen fel valami (ekkor esetleg állhat NULL-érték is bizonyos helyeken, ami jelentheti azt, hogy nincs megfelelő érték, vagy hogy van, de nem ismert),
    - pontosan egyet vehessen fel (pl. kulcsattribútum nem lehet NULL).

  Hogy melyik megközelítés van, az rendszerfüggő.
  A NULL érték megjelenítésére eszközök az ODL-ben:

    - értelmezési tartományon kívüli érték (film hossza -1),

Megszorítások megadása ODL-ben

- *Egyértékűség az ODL-ben*:
  - Kulcs-szerű megszorítás jól leírható (lásd előbb)
  - Az attribútumok és a kapcsolatok többességének szabályozására: a kollekcióoperátorok használata/nem használata. Így előírható, hogy egy attribútum/kapcsolat csak egy értéket vehessen fel.
  - Egyértékűséget kétféleképpen is lehet érteni:
    - legfeljebb egy értéken vehessen fel valami (ekkor esetleg állhat NULL-érték is bizonyos helyeken, ami jelentheti azt, hogy nincs megfelelő érték, vagy hogy van, de nem ismert),
    - pontosan egyet vehessen fel (pl. kulcsattribútum nem lehet NULL).

    Hogy melyik megközelítés van, az rendszerfüggő.
    A NULL érték megjelenítésére eszközök az ODL-ben:

    - értelmezési tartományon kívüli érték (film hossza -1),
    - felsorolástípusnál külön megadva (enum szalagfajta {ff, sz, null}).

- *Hivatkozási épség*:
  Cél, hogy ha valahol van valamire hivatkozás, akkor az létezzen is. Pl. ha a Filmnél van mutató egy Stúdióra, mint gyártóra, akkor legyen olyan stúdió a Stúdió osztályban.

- *Hivatkozási épség*:
  Cél, hogy ha valahol van valamire hivatkozás, akkor az létezzen is. Pl. ha a Filmnél van mutató egy Stúdióra, mint gyártóra, akkor legyen olyan stúdió a Stúdió osztályban.
  Erre figyelni bonyolult:
  - ne lehessen úgy filmet felvenni, hogy nincs hozzá stúdió

- *Hivatkozási épség*:
  Cél, hogy ha valahol van valamire hivatkozás, akkor az létezzen is. Pl. ha a Filmnél van mutató egy Stúdióra, mint gyártóra, akkor legyen olyan stúdió a Stúdió osztályban.
  Erre figyelni bonyolult:
  - ne lehessen úgy filmet felvenni, hogy nincs hozzá stúdió
  - ne lehessen ész nélkül stúdiót törölni

- *Hivatkozási épség*:
  Cél, hogy ha valahol van valamire hivatkozás, akkor az létezzen is. Pl. ha a Filmnél van mutató egy Stúdióra, mint gyártóra, akkor legyen olyan stúdió a Stúdió osztályban.
  Erre figyelni bonyolult:
  - ne lehessen úgy filmet felvenni, hogy nincs hozzá stúdió
  - ne lehessen ész nélkül stúdiót törölni

  Az ODL az egész hivatkozási épség kérdést a megvalósítás szintjére tolja át.

- *Hivatkozási épség*:
  Cél, hogy ha valahol van valamire hivatkozás, akkor az létezzen is. Pl. ha a Filmnél van mutató egy Stúdióra, mint gyártóra, akkor legyen olyan stúdió a Stúdió osztályban.
  Erre figyelni bonyolult:
  - ne lehessen úgy filmet felvenni, hogy nincs hozzá stúdió
  - ne lehessen ész nélkül stúdiót törölni

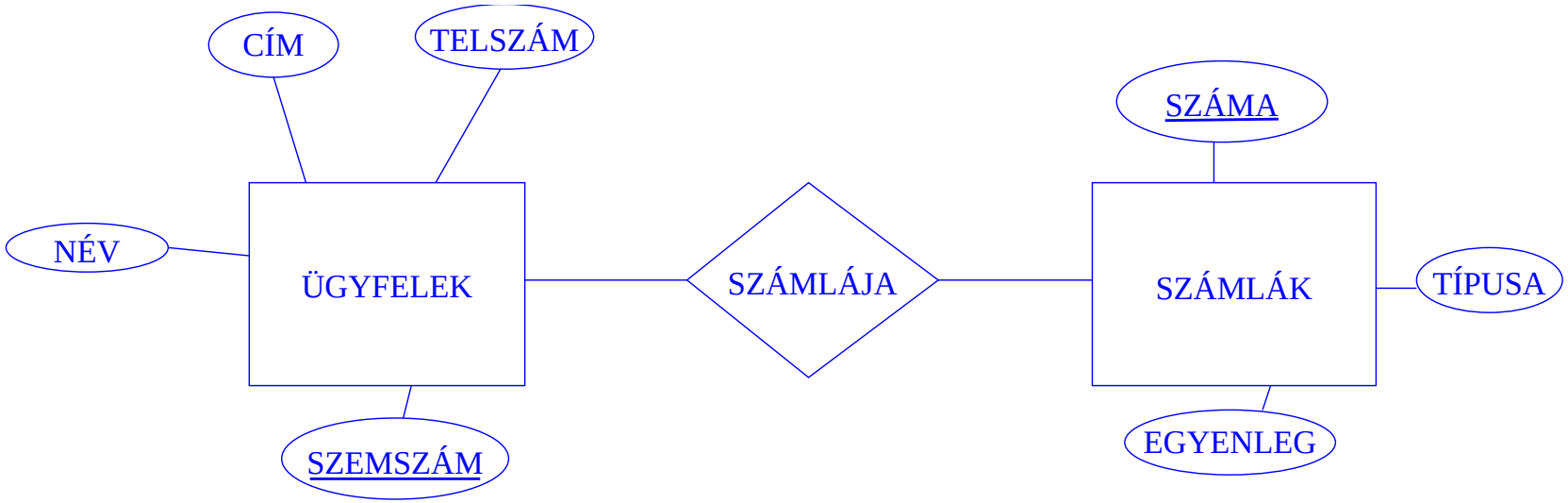
  Az ODL az egész hivatkozási épség kérdést a megvalósítás szintjére tolja át.
- *Értelmezési tartomány megszorítása és egyéb megkötések*:
  Az értelmezési tartomány megszorítására a típusok vannak, további szűkítést nem támogat. A kapcsolat fokát lehet korlátozni az Array kollekcióoperátor használatával (Array<Színész, 10> esetén csak 10 színészt tartunk nyilván).

Megszorítások E/K modellben

- *Kulcsok*:
  egy kulcsot aláhúzással jelölünk (a kulcsba tartozó attribútumokat aláhúzzuk), a többi kulcsot az ábrán nem lehet jelölni, ezeket szövegesen mellékeljük.

Megszorítások E/K modellben

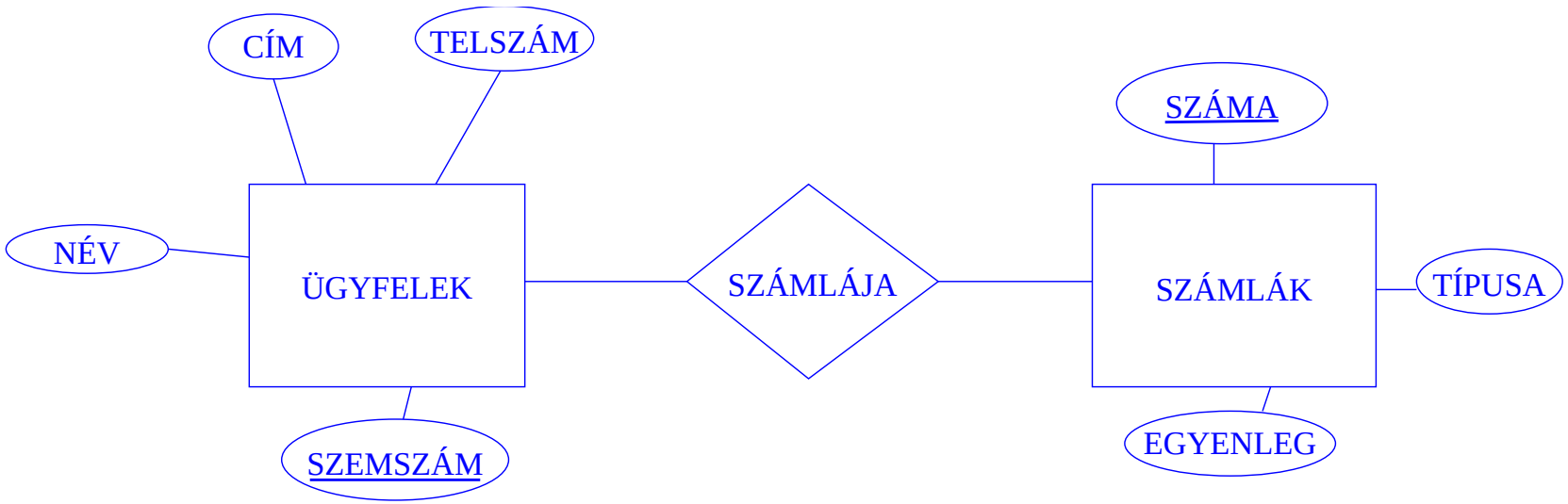
- *Kulcsok*:
  egy kulcsot aláhúzással jelölünk (a kulcsba tartozó attribútumokat aláhúzzuk), a többi kulcsot az ábrán nem lehet jelölni, ezeket szövegesen mellékeljük.



- *Egyértékűség*:
  - egyszerű attribútumok használata $\Longrightarrow$ minden attribútum egyértékű az E/K modellben (általában lehet NULL-érték is, ha mégsem, akkor írásban jelezhető)

Megszorítások E/K modellben

- *Kulcsok*:
  egy kulcsot aláhúzással jelölünk (a kulcsba tartozó attribútumokat aláhúzzuk), a többi kulcsot az ábrán nem lehet jelölni, ezeket szövegesen mellékeljük.



- *Egyértékűség*:
  - egyszerű attribútumok használata $\Longrightarrow$ minden attribútum egyértékű az E/K modellben (általában lehet NULL-érték is, ha mégsem, akkor írásban jelezhető)
  - kapcsolatnál: nyilakkal jelezhető, ha valamerre „egy” a kapcsolat

- *Hivatkozási épség*:
  lehet a rajzon jelezni, ha egy kapcsolatnál azt szeretnénk, hogy pontosan egy egyed tartozzon egy kiválasztott egyedhez. Ilyenkor kerek nyilat használunk:

  Ebben az esetben minden filmhez pontosan egy stúdiónak kell tartoznia.

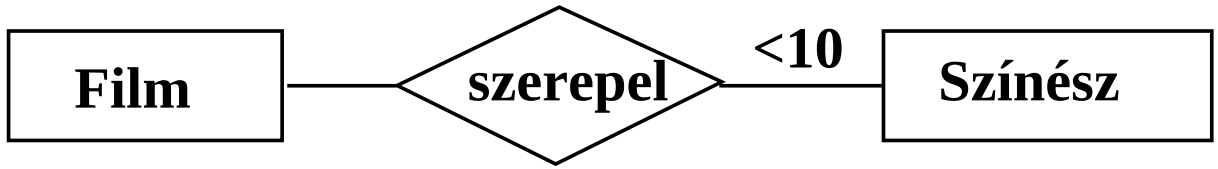
- *Hivatkozási épség*:
  lehet a rajzon jelezni, ha egy kapcsolatnál azt szeretnénk, hogy pontosan egy egyed tartozzon egy kiválasztott egyedhez. Ilyenkor kerek nyilat használunk:

  Ebben az esetben minden filmhez pontosan egy stúdiónak kell tartoznia.
- *Értelmezési tartományra vonatkozó megkötések és egyéb megszorítások*:
  Értelmezési tartomány: típussal.

- *Hivatkozási épség*:
  lehet a rajzon jelezni, ha egy kapcsolatnál azt szeretnénk, hogy pontosan egy egyed tartozzon egy kiválasztott egyedhez. Ilyenkor kerek nyilat használunk:

  Ebben az esetben minden filmhez pontosan egy stúdiónak kell tartoznia.

- *Értelmezési tartományra vonatkozó megkötések és egyéb megszorítások*:
  Értelmezési tartomány: típussal.
  Egyéb: kapcsolat fokát lehet itt is korlátozni, pl:



  Ekkor egy filmhez 10-nél kevesebb színészt rendelünk.

Adatbázisok elmélete

Gyenge egyedhalmazok

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

4. előadás

Gyenge egyedhalmazok

Az E/K modell sajátossága. Egy egyedhalmaz akkor gyenge egyedhalmaz, ha az egyedeit nem azonosítják az attribútumai, csak a kapcsolatokkal együtt. (ODL-nél nincs ez a dolog, mert ott az egyedi OID mindig azonosít.)

Gyenge egyedhalmazok

Az E/K modell sajátossága. Egy egyedhalmaz akkor gyenge egyedhalmaz, ha az egyedeit nem azonosítják az attribútumai, csak a kapcsolatokkal együtt. (ODL-nél nincs ez a dolog, mert ott az egyedi OID mindig azonosít.)
Jelölés: dupla téglalap az egyedhalmaznak és dupla rombusz azoknak a kapcsolatoknak, amiken keresztül megy az azonosítás.

Gyenge egyedhalmazok

Az E/K modell sajátossága. Egy egyedhalmaz akkor gyenge egyedhalmaz, ha az egyedeit nem azonosítják az attribútumai, csak a kapcsolatokkal együtt. (ODL-nél nincs ez a dolog, mert ott az egyedi OID mindig azonosít.)
Jelölés: dupla téglalap az egyedhalmaznak és dupla rombusz azoknak a kapcsolatoknak, amiken keresztül megy az azonosítás.
A gyenge egyedhalmaznál az aláhúzott attribútumok belekerülnek a gyenge egyedhalmaz kulcsába, de még más attribútumok is hozzájönnek ehhez: azok, amik a duplarombuszos kapcsolat(ok) végén álló egyedhalmaz(ok) kulcsai.

Példák:

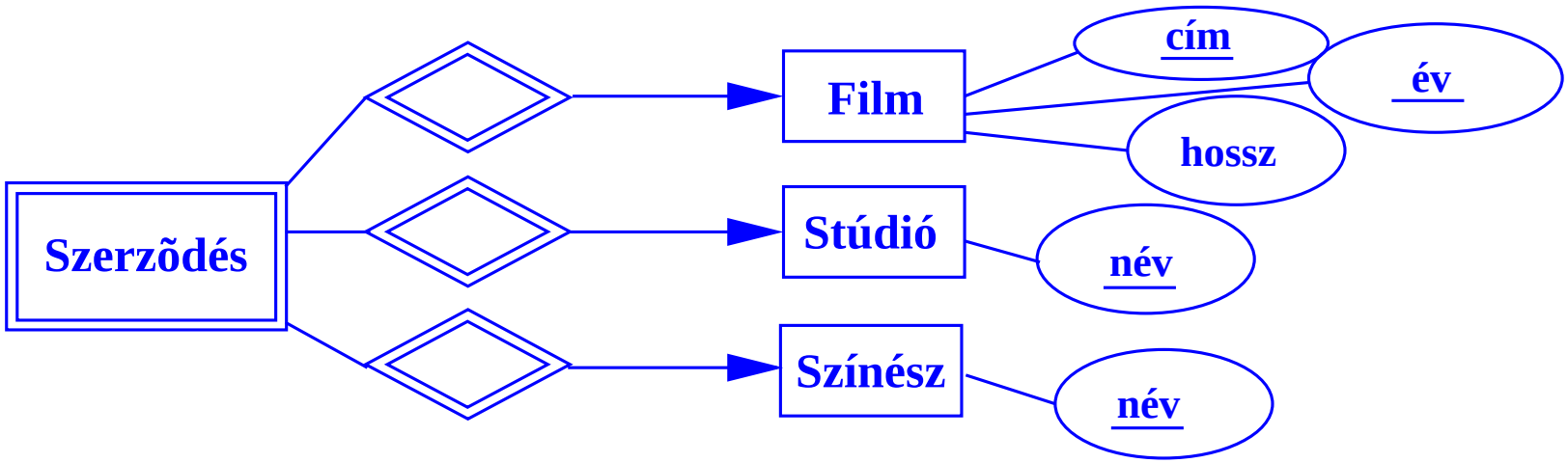
- Amikor a többágú kapcsolatot binárissá írtuk át, akkor olyan egyedhalmaz keletkezik (a kapcsolatból), aminek általában nincs is attribútuma, ezért ennek az egyedhalmaznak az egyedeit csak a kapcsolatokon át lehet azonosítani.

Példák:

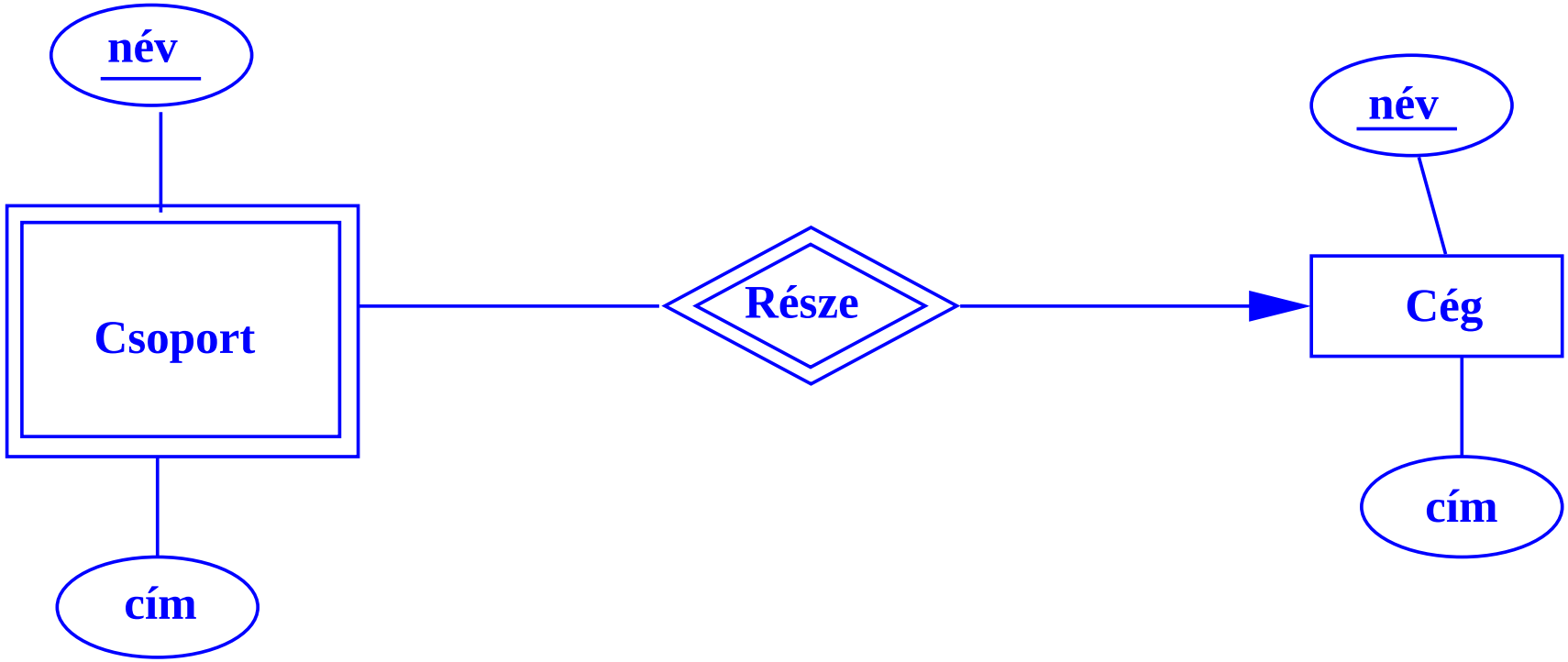
- Amikor a többágú kapcsolatot binárissá írtuk át, akkor olyan egyedhalmaz keletkezik (a kapcsolatból), aminek általában nincs is attribútuma, ezért ennek az egyedhalmaznak az egyedeit csak a kapcsolatokon át lehet azonosítani.
  A filmes példa esetén a Szerződés egyedhalmaz egyedeit a kapcsolódó egyedhalmazok (Film, Színész, Stúdió) kulcsattribútumai azonosítják: film címe, gyártási éve, színész neve, stúdió neve. Ha ezek adottak, akkor már csak egy szerződés lehet, ami ezekre vonatkozik.

Példák:

- Amikor a többágú kapcsolatot binárissá írtuk át, akkor olyan egyedhalmaz keletkezik (a kapcsolatból), aminek általában nincs is attribútuma, ezért ennek az egyedhalmaznak az egyedeit csak a kapcsolatokon át lehet azonosítani.
  A filmes példa esetén a Szerződés egyedhalmaz egyedeit a kapcsolódó egyedhalmazok (Film, Színész, Stúdió) kulcsattribútumai azonosítják: film címe, gyártási éve, színész neve, stúdió neve. Ha ezek adottak, akkor már csak egy szerződés lehet, ami ezekre vonatkozik.



- Ebben a példában a csoport neve még önmagában nem kulcs (sok cégnél lehet pl. HR csoport), sőt a címmel együtt sem feltétlenül azonosít egy csoportot, de ha a kapcsolaton keresztül a céget is bevesszük az azonosításba, úgy már egyértelmű lesz, hogy melyik csoportról beszélünk.



Követelmények az azonosító kapcsolatra

A gyenge egyedhalmaz kulcsában benne lehetnek saját attribútumai (mint az előbb a Csoport neve) és biztosan vannak benne olyan attribútumok, amiket duplarombuszos kapcsolat(ok)on keresztül szerez.

Követelmények az azonosító kapcsolatra

A gyenge egyedhalmaz kulcsában benne lehetnek saját attribútumai (mint az előbb a Csoport neve) és biztosan vannak benne olyan attribútumok, amiket duplarombuszos kapcsolat(ok)on keresztül szerez.
Követelmények ezekre a kapcsolatokra:

1. Ha az E gyenge egyedhalmaz kulcsattribútumot szerez egy F egyedhalmaztól az R kapcsolaton át, akkor R legyen több-egy E-ből F-be. (Így egy E-belihez egyértelműen tartozik egy F-beli).

Követelmények az azonosító kapcsolatra

A gyenge egyedhalmaz kulcsában benne lehetnek saját attribútumai (mint az előbb a Csoport neve) és biztosan vannak benne olyan attribútumok, amiket duplarombuszos kapcsolat(ok)on keresztül szerez.
Követelmények ezekre a kapcsolatokra:

1. Ha az E gyenge egyedhalmaz kulcsattribútumot szerez egy F egyedhalmaztól az R kapcsolaton át, akkor R legyen több-egy E-ből F-be. (Így egy E-belihez egyértelműen tartozik egy F-beli).
2. Egy attribútum pontosan akkor kerül bele az E gyenge egyedhalmaz kulcsába, ha benne van az F egyedhalmaz kulcsában is.

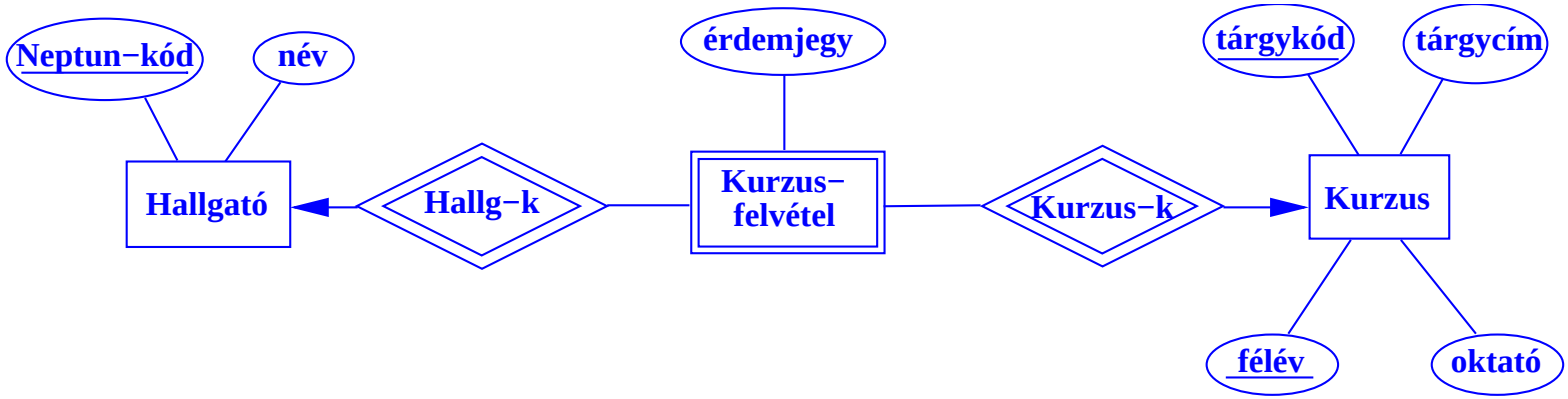
Megjegyzés: természetesen F is lehet gyenge egyedhalmaz.

Példa

Tervezzen E/K diagrammot egy egyetemi nyilvántartáshoz, ahol hallgatókat és az általuk szerzett jegyeket tartjuk nyilván. Vegyünk három egyedhalmazt: hallgató, kurzus, kurzusfelvétel (ez utóbbi kapcsoló egyedhalmaz a hallgatók és kurzusok között, ennél reprezentáljuk a kapott érdemjegyet is). Adjuk meg ezt E/K diagrammal, jelöljük a gyenge egyedhalmazokat és a kulcsokat is.

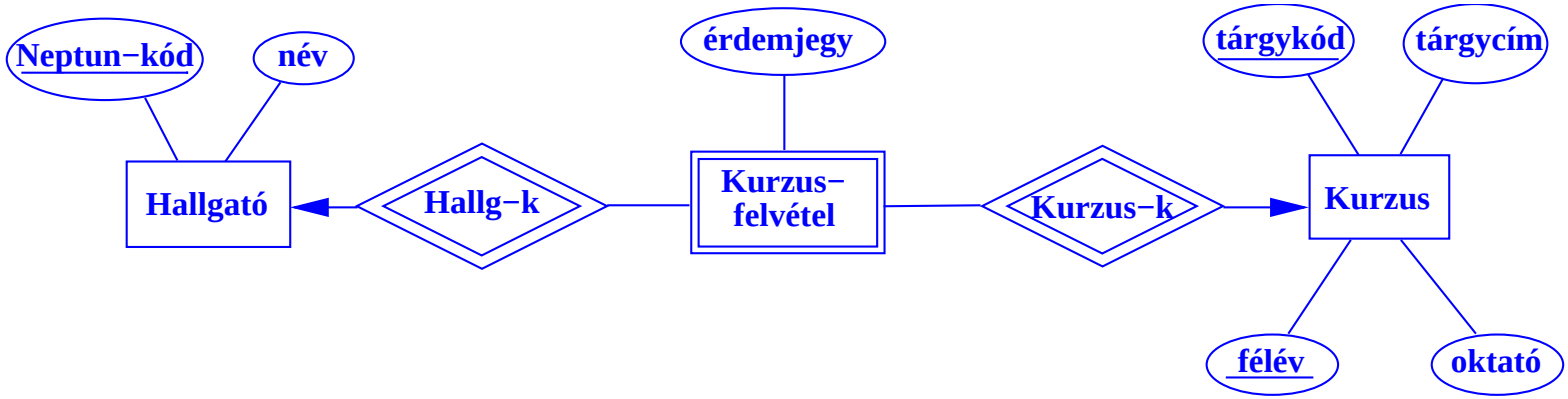
Példa

Tervezzen E/K diagrammot egy egyetemi nyilvántartáshoz, ahol hallgatókat és az általuk szerzett jegyeket tartjuk nyilván. Vegyünk három egyedhalmazt: hallgató, kurzus, kurzusfelvétel (ez utóbbi kapcsoló egyedhalmaz a hallgatók és kurzusok között, ennél reprezentáljuk a kapott érdemjegyet is). Adjuk meg ezt E/K diagrammal, jelöljük a gyenge egyedhalmazokat és a kulcsokat is.



Példa

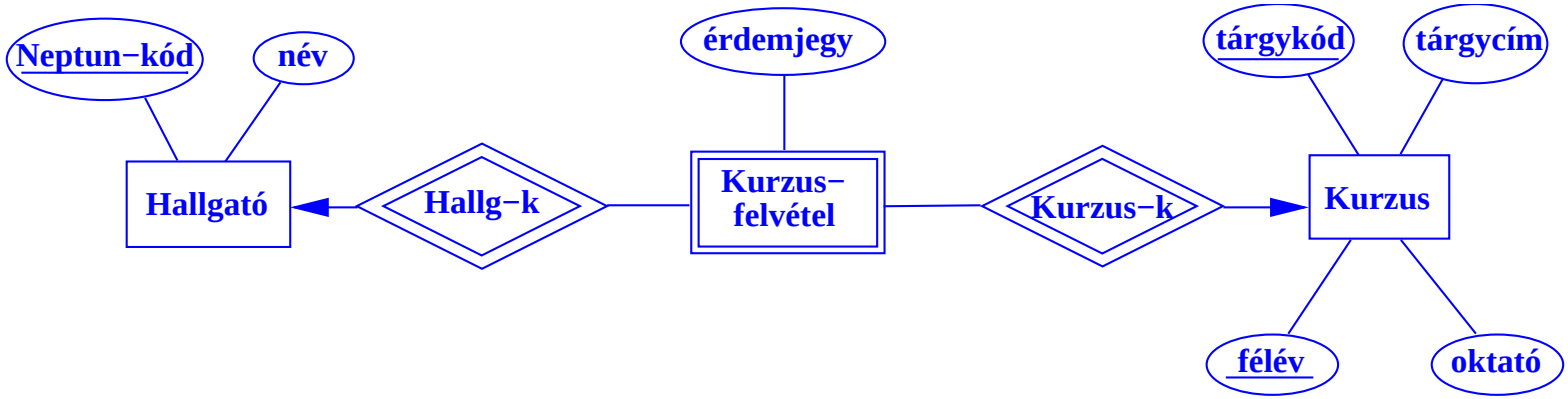
Tervezzen E/K diagrammot egy egyetemi nyilvántartáshoz, ahol hallgatókat és az általuk szerzett jegyeket tartjuk nyilván. Vegyünk három egyedhalmazt: hallgató, kurzus, kurzusfelvétel (ez utóbbi kapcsoló egyedhalmaz a hallgatók és kurzusok között, ennél reprezentáljuk a kapott érdemjegyet is). Adjuk meg ezt E/K diagrammal, jelöljük a gyenge egyedhalmazokat és a kulcsokat is.



Döntsük el, hogy az érdemjegy része-e a kurzusfelvételt reprezentáló egyedhalmaz kulcsának?

Példa

Tervezzen E/K diagrammot egy egyetemi nyilvántartáshoz, ahol hallgatókat és az általuk szerzett jegyeket tartjuk nyilván. Vegyünk három egyedhalmazt: hallgató, kurzus, kurzusfelvétel (ez utóbbi kapcsoló egyedhalmaz a hallgatók és kurzusok között, ennél reprezentáljuk a kapott érdemjegyet is). Adjuk meg ezt E/K diagrammal, jelöljük a gyenge egyedhalmazokat és a kulcsokat is.



Döntsük el, hogy az érdemjegy része-e a kurzusfelvételt reprezentáló egyedhalmaz kulcsának?

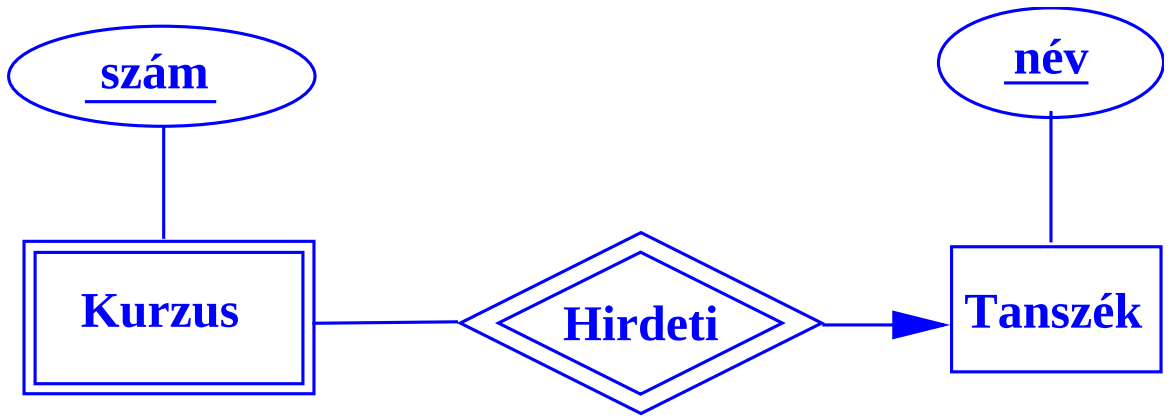
Az érdemjegy nem része a kurzusfelvétel egyedhalmaz kulcsának, ezen egyedhalmaz kulcsa a két kapcsolaton keresztül jön: a hallgatótól a neptun-kód, a tárgytól meg a tárgykód és a félév.

Példa

Tervezzen E/K diagrammot a következőre és jelölje a rajzon a kulcsokat és a gyenge egyedhalmazokat:
Egyedhalmazok: Kurzusok, Tanszékek. Egy kurzust egy tanszék hirdet meg, de azt csak egy számmal azonosítja. Különböző tanszékek adhatják ugyanazt a számot a kurzusuknak, de egy tanszék tárgyai mind különböző számot kapnak.

Példa

Tervezzen E/K diagrammot a következőre és jelölje a rajzon a kulcsokat és a gyenge egyedhalmazokat:
Egyedhalmazok: Kurzusok, Tanszékek. Egy kurzust egy tanszék hirdet meg, de azt csak egy számmal azonosítja. Különböző tanszékek adhatják ugyanazt a számot a kurzusuknak, de egy tanszék tárgyai mind különböző számot kapnak.

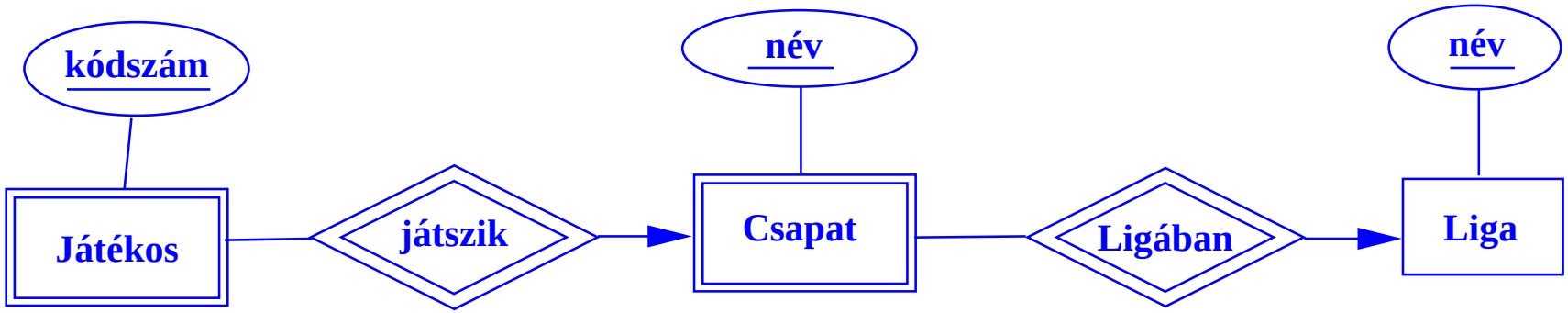


Példa

Tervezzen E/K diagrammot a következőre és jelölje a rajzon a kulcsokat és a gyenge egyedhalmazokat:
Egyedhalmazok: Ligák, Csapatok, Játékosok. A Ligák nevei egyediek, a Csapatoké egy ligán belül különbözik, de különböző ligán belül lehetnek azonos nevű csapatok. Egy csapaton belül nincsenek azonos kódszámú játékosok, de különböző csapatokban lehetnek ilyenek.

Példa

Tervezzen E/K diagrammot a következőre és jelölje a rajzon a kulcsokat és a gyenge egyedhalmazokat:
Egyedhalmazok: Ligák, Csapatok, Játékosok. A Ligák nevei egyediek, a Csapatoké egy ligán belül különbözik, de különböző ligán belül lehetnek azonos nevű csapatok. Egy csapaton belül nincsenek azonos kódszámú játékosok, de különböző csapatokban lehetnek ilyenek.



Miért vannak gyenge egyedhalmazok?

- Maguktól keletkeznek, amikor többágú kapcsolatot írunk át binárissá.

Miért vannak gyenge egyedhalmazok?

- Maguktól keletkeznek, amikor többágú kapcsolatot írunk át binárissá.
- A redundancia elkerülése céljából. (Minek a cég nevét minden csoportnál külön felvenni, elég ha egyszer felírjuk és a kapcsolatból derítjük ki.)

Miért vannak gyenge egyedhalmazok?

- Maguktól keletkeznek, amikor többágú kapcsolatot írunk át binárissá.
- A redundancia elkerülése céljából. (Minek a cég nevét minden csoportnál külön felvenni, elég ha egyszer felírjuk és a kapcsolatból derítjük ki.)

A redundancia elkerülése nem csak az E/K modellben fontos, ez minden megközelítésben lényeges, hisz a redundancia bajok forrása.

Miért vannak gyenge egyedhalmazok?

- Maguktól keletkeznek, amikor többágú kapcsolatot írunk át binárissá.
- A redundancia elkerülése céljából. (Minek a cég nevét minden csoportnál külön felvenni, elég ha egyszer felírjuk és a kapcsolatból derítjük ki.)

A redundancia elkerülése nem csak az E/K modellben fontos, ez minden megközelítésben lényeges, hisz a redundancia bajok forrása.

- Nehéz konzisztens állapotban tartani a DB-t, ha ugyanaz az infó ezer helyen van beírva.

Miért vannak gyenge egyedhalmazok?

- Maguktól keletkeznek, amikor többágú kapcsolatot írunk át binárissá.
- A redundancia elkerülése céljából. (Minek a cég nevét minden csoportnál külön felvenni, elég ha egyszer felírjuk és a kapcsolatból derítjük ki.)

A redundancia elkerülése nem csak az E/K modellben fontos, ez minden megközelítésben lényeges, hisz a redundancia bajok forrása.

- Nehéz konzisztens állapotban tartani a DB-t, ha ugyanaz az infó ezer helyen van beírva.
- Nem lesz elég egyszerű a séma, nehéz lesz átlátni, hogy mi az ami ugyanaz, csak sokszor tároljuk és mi valóban más infó.

Miért vannak gyenge egyedhalmazok?

- Maguktól keletkeznek, amikor többágú kapcsolatot írunk át binárissá.
- A redundancia elkerülése céljából. (Minek a cég nevét minden csoportnál külön felvenni, elég ha egyszer felírjuk és a kapcsolatból derítjük ki.)

A redundancia elkerülése nem csak az E/K modellben fontos, ez minden megközelítésben lényeges, hisz a redundancia bajok forrása.

- Nehéz konzisztens állapotban tartani a DB-t, ha ugyanaz az infó ezer helyen van beírva.
- Nem lesz elég egyszerű a séma, nehéz lesz átlátni, hogy mi az ami ugyanaz, csak sokszor tároljuk és mi valóban más infó.
- Helyprobléma (ez egyre kevésbé van).

Miért vannak gyenge egyedhalmazok?

- Maguktól keletkeznek, amikor többágú kapcsolatot írunk át binárissá.
- A redundancia elkerülése céljából. (Minek a cég nevét minden csoportnál külön felvenni, elég ha egyszer felírjuk és a kapcsolatból derítjük ki.)

A redundancia elkerülése nem csak az E/K modellben fontos, ez minden megközelítésben lényeges, hisz a redundancia bajok forrása.

- Nehéz konzisztens állapotban tartani a DB-t, ha ugyanaz az infó ezer helyen van beírva.
- Nem lesz elég egyszerű a séma, nehéz lesz átlátni, hogy mi az ami ugyanaz, csak sokszor tároljuk és mi valóban más infó.
- Helyprobléma (ez egyre kevésbé van).

Ezek miatt törekszünk a redundancia kiküszöbölésére, de persze nem kell mindent kiirtani, hisz a világ is redundáns.

Tervezési alapelvek

1. *Valósághű modellezés*: megragadni a lényeget, megfelelő adatelemeket választani, megfelelő kapcsolatokat (természetesek legyenek, de néha kellenek mesterséges, technikai egyedhalmazok, osztályok is).

Tervezési alapelvek

1. *Valósághű modellezés*: megragadni a lényeget, megfelelő adatelemeket választani, megfelelő kapcsolatokat (természetesek legyenek, de néha kellenek mesterséges, technikai egyedhalmazok, osztályok is).
2. *Redundancia kerülése*: észszerű mértékben. Ezt majd a relációs modell nagyon jól megoldja, de azért már a tervezéskor is jó erre figyelni.

Tervezési alapelvek

1. *Valósághű modellezés*: megragadni a lényeget, megfelelő adatelemeket választani, megfelelő kapcsolatokat (természetesek legyenek, de néha kellenek mesterséges, technikai egyedhalmazok, osztályok is).
2. *Redundancia kerülése*: észszerű mértékben. Ezt majd a relációs modell nagyon jól megoldja, de azért már a tervezéskor is jó erre figyelni.
3. *Egyszerűség*: csak az legyen a sémában, aminek lennie kell, minél egyszerűbb szerkezetben.

Tervezési alapelvek

1. *Valósághű modellezés*: megragadni a lényeget, megfelelő adatelemeket választani, megfelelő kapcsolatokat (természetesek legyenek, de néha kellenek mesterséges, technikai egyedhalmazok, osztályok is).
2. *Redundancia kerülése*: észszerű mértékben. Ezt majd a relációs modell nagyon jól megoldja, de azért már a tervezéskor is jó erre figyelni.
3. *Egyszerűség*: csak az legyen a sémában, aminek lennie kell, minél egyszerűbb szerkezetben.
4. *Megfelelő (típusú, összetettségű) adatelemek választása*: jól döntsünk, hogy mi legyen attribútum, mi inkább kapcsolat, illetve esetleg külön osztály/egyedhalmaz. Az attribútumot egyszerűbb implementálni, de néha átláthatóbb egy külön egyedhalmaz.

Tervezési alapelvek

1. *Valósághű modellezés*: megragadni a lényeget, megfelelő adatelemeket választani, megfelelő kapcsolatokat (természetesek legyenek, de néha kellenek mesterséges, technikai egyedhalmazok, osztályok is).
2. *Redundancia kerülése*: észszerű mértékben. Ezt majd a relációs modell nagyon jól megoldja, de azért már a tervezéskor is jó erre figyelni.
3. *Egyszerűség*: csak az legyen a sémában, aminek lennie kell, minél egyszerűbb szerkezetben.
4. *Megfelelő (típusú, összetettségű) adatelemek választása*: jól döntsünk, hogy mi legyen attribútum, mi inkább kapcsolat, illetve esetleg külön osztály/egyedhalmaz. Az attribútumot egyszerűbb implementálni, de néha átláthatóbb egy külön egyedhalmaz.

Általános elvek:

- ha egy egyedhalmaznak csak egy attribútuma lenne $\Longrightarrow$ nem érdemes külön venni, ha összetettebb, akkor legyen külön.

Általános elvek:

- ha egy egyedhalmaznak csak egy attribútuma lenne $\Longrightarrow$ nem érdemes külön venni, ha összetettebb, akkor legyen külön.
- ha egy infót magában nem akarunk megőrizni, csak valamihez kapcsoltan $\Longrightarrow$ lehet csak attribútum (pl. ha a stúdiók csak annyiban érdekelnek minket, hogy melyik filmet ki gyártja, akkor nem kell külön Stúdió egyedhalmaz)

Általános elvek:

- ha egy egyedhalmaznak csak egy attribútuma lenne $\Longrightarrow$ nem érdemes külön venni, ha összetettebb, akkor legyen külön.
- ha egy infót magában nem akarunk megőrizni, csak valamihez kapcsoltan $\Longrightarrow$ lehet csak attribútum (pl. ha a stúdiók csak annyiban érdekelnek minket, hogy melyik filmet ki gyártja, akkor nem kell külön Stúdió egyedhalmaz)

Ez mind a modellezéskor dől el, aszerint, hogy milyen sémát akarunk.

Adatbázisok elmélete

Relációs adatmodell, relációs algebra

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

5. előadás

Régebbi adatmodellek

- *Hálós adatmodell*: szemléletében hasonlít az objektumosra, de itt sokkal jobban közelíti a terv a fizikai megvalósítást (pl. az attribútumok megadásánál rögtön rendelkezünk a tárolás módjáról is). Lekérdezés, módosítás csak a tárolás pontos ismeretében lehetséges $\Longrightarrow$ nehézkesebb mint a relációs modell használata.

Régebbi adatmodellek

- *Hálós adatmodell*: szemléletében hasonlít az objektumosra, de itt sokkal jobban közelíti a terv a fizikai megvalósítást (pl. az attribútumok megadásánál rögtön rendelkezünk a tárolás módjáról is). Lekérdezés, módosítás csak a tárolás pontos ismeretében lehetséges $\Longrightarrow$ nehézkesebb mint a relációs modell használata.
- *Hierarchikus adatmodell*: az első, korai rendszerek hierarchikussága miatt szervesen alakult ki. Akkor jó, ha az adatok, vagy a tárolás hierarchikus szerkezetű. Itt is ismerni kell a fizikai megvalósítást a kérdezéshez/módosításhoz.

Relációs adatmodell

Jelenleg ez a legelterjedtebb, szinte minden DBMS ezen az elven működik.
Ennek okai:

- jól lehet benne modellezni, a modell után pedig könnyű a konkrét sémát megvalósítani

Relációs adatmodell

Jelenleg ez a legelterjedtebb, szinte minden DBMS ezen az elven működik.
Ennek okai:

- jól lehet benne modellezni, a modell után pedig könnyű a konkrét sémát megvalósítani
- nem kell ismerni a fizikai megvalósítást a lekérdezéshez, módosításhoz

Relációs adatmodell

Jelenleg ez a legelterjedtebb, szinte minden DBMS ezen az elven működik.
Ennek okai:

- jól lehet benne modellezni, a modell után pedig könnyű a konkrét sémát megvalósítani
- nem kell ismerni a fizikai megvalósítást a lekérdezéshez, módosításhoz
- a logikai tervezésnek nagy, szép matematikai eszköztára van, ami segíti az egyszerű séma létrehozását

Relációs adatmodell

Mit fogunk róla tanulni?

Relációs adatmodell

Mit fogunk róla tanulni?

1. elvi keret (alapfogalmak, alapműveletek)

Relációs adatmodell

Mit fogunk róla tanulni?

1. elvi keret (alapfogalmak, alapműveletek)
2. konkrét nyelvek (ISBL, QBE, QUELL, SQL, sémadefinícióra, adatmódosításra és lekérdezésre)

Relációs adatmodell

Mit fogunk róla tanulni?

1. elvi keret (alapfogalmak, alapműveletek)
2. konkrét nyelvek (ISBL, QBE, QUELL, SQL, sémadefinícióra, adatmódosításra és lekérdezésre)
3. tervezés (minél jobb séma kialakítása, matematikai elmélet)

Relációs adatmodell

Mit fogunk róla tanulni?

1. elvi keret (alapfogalmak, alapműveletek)
2. konkrét nyelvek (ISBL, QBE, QUELL, SQL, sémadefinícióra, adatmódosításra és lekérdezésre)
3. tervezés (minél jobb séma kialakítása, matematikai elmélet)

Egyetlen alapfogalom (nincs külön egyedhalmaz és kapcsolat): reláció.

A reláció definíciója

1. Gondolhatunk rá úgy, mint egy síkbeli táblázatra:

R_1	A_1	A_2
	1	y
	1	z
	3	y

R_2	A_1	A_2
	2	y
	1	z

Itt R_1 a reláció neve, A_1 és A_2 az attribútumok nevei, a sorok pedig a reláció elemei. Az oszlopokban levő értékek az attribútumokhoz tartozó értékkészletből kerülnek ki.

2. Tekinthetjük egy Descartes-szorzat részhalmazának is a relációt:
$A_1, A_2, \ldots, A_n$ tetszőleges halmazok (attribútumok)

$$R \subseteq A_1 \times \cdots \times A_n$$

$\Longrightarrow$ Minden sor csak egyszer szerepel

2. Tekinthetjük egy Descartes-szorzat részhalmazának is a relációt:
$A_1, A_2, \ldots, A_n$ tetszőleges halmazok (attribútumok)

$$R \subseteq A_1 \times \cdots \times A_n$$

$\Longrightarrow$ Minden sor csak egyszer szerepel
$\Longrightarrow$ a sorok sorrendje lényegtelen.

2. Tekinthetjük egy Descartes-szorzat részhalmazának is a relációt:
$A_1, A_2, \ldots, A_n$ tetszőleges halmazok (attribútumok)

$$R \subseteq A_1 \times \cdots \times A_n$$

$\Longrightarrow$ Minden sor csak egyszer szerepel
$\Longrightarrow$ a sorok sorrendje lényegtelen.
Példa: $A_1 = \{1, 2, 3\}, A_2 = \{x, y, z\}$

2. Tekinthetjük egy Descartes-szorzat részhalmazának is a relációt:
$A_1, A_2, \ldots, A_n$ tetszőleges halmazok (attribútumok)

$$R \subseteq A_1 \times \cdots \times A_n$$

$\Longrightarrow$ Minden sor csak egyszer szerepel
$\Longrightarrow$ a sorok sorrendje lényegtelen.
Példa: $A_1 = \{1, 2, 3\}, A_2 = \{x, y, z\}$
$R_1 = \{\{1, y\}, \{1, z\}, \{3, z\}\}$

2. Tekinthetjük egy Descartes-szorzat részhalmazának is a relációt:
$A_1, A_2, \ldots, A_n$ tetszőleges halmazok (attribútumok)

$$R \subseteq A_1 \times \cdots \times A_n$$

$\Longrightarrow$ Minden sor csak egyszer szerepel
$\Longrightarrow$ a sorok sorrendje lényegtelen.
Példa: $A_1 = \{1, 2, 3\}, A_2 = \{x, y, z\}$
$R_1 = \{\{1, y\}, \{1, z\}, \{3, z\}\}$
$R_2 = \{\{2, y\}, \{1, z\}\}$

2. Tekinthetjük egy Descartes-szorzat részhalmazának is a relációt:
$A_1, A_2, \ldots, A_n$ tetszőleges halmazok (attribútumok)

$$R \subseteq A_1 \times \cdots \times A_n$$

$\Longrightarrow$ Minden sor csak egyszer szerepel
$\Longrightarrow$ a sorok sorrendje lényegtelen.
Példa: $A_1 = \{1, 2, 3\}, A_2 = \{x, y, z\}$
$R_1 = \{\{1, y\}, \{1, z\}, \{3, z\}\}$
$R_2 = \{\{2, y\}, \{1, z\}\}$
De R elemeit tekinthetjük halmazoknak is, nem rendezett n-eseknek.

2. Tekinthetjük egy Descartes-szorzat részhalmazának is a relációt:
$A_1, A_2, \ldots, A_n$ tetszőleges halmazok (attribútumok)

$$R \subseteq A_1 \times \cdots \times A_n$$

$\Longrightarrow$ Minden sor csak egyszer szerepel
$\Longrightarrow$ a sorok sorrendje lényegtelen.
Példa: $A_1 = \{1, 2, 3\}, A_2 = \{x, y, z\}$
$R_1 = \{\{1, y\}, \{1, z\}, \{3, z\}\}$
$R_2 = \{\{2, y\}, \{1, z\}\}$
De R elemeit tekinthetjük halmazoknak is, nem rendezett n-eseknek.
Ekkor az attribútumok sorrendje is mindegy.

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{attribútumok\} \to \{attr.\ értékkészlete\}$

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{\textit{attribútumok}\} \to \{\textit{attr. értékkészlete}\}$
Egy R reláció ilyen függvények halmaza.

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{attribútumok\} \to \{attr.\ értékkészlete\}$
Egy R reláció ilyen függvények halmaza.

Így tényleg nem számít a sorrend, se a sorok között, se az attribútumok között.

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{attribútumok\} \to \{attr.\ értékkészlete\}$
Egy R reláció ilyen függvények halmaza.

Így tényleg nem számít a sorrend, se a sorok között, se az attribútumok között.
Nincs két azonos sor.

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{attribútumok\} \to \{attr.\ értékkészlete\}$
Egy R reláció ilyen függvények halmaza.

Így tényleg nem számít a sorrend, se a sorok között, se az attribútumok között.
Nincs két azonos sor.
Például:
R_1-ben: 1. sor: $A_1 \to 1$; $A_2 \to$ y;

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{attribútumok\} \to \{attr.\ értékkészlete\}$
Egy R reláció ilyen függvények halmaza.

Így tényleg nem számít a sorrend, se a sorok között, se az attribútumok között.
Nincs két azonos sor.
Például:
R_1-ben: 1. sor: $A_1 \to 1$; $A_2 \to$ y;

Jelölés:

Definíció

Relációs séma: $R(A_1, \ldots, A_n)$, *ahol R a reláció neve, az* A_i*-k pedig az attribútumok nevei.*

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{attribútumok\} \to \{attr.\ értékkészlete\}$
Egy R reláció ilyen függvények halmaza.

Így tényleg nem számít a sorrend, se a sorok között, se az attribútumok között.
Nincs két azonos sor.
Például:
R_1-ben: 1. sor: $A_1 \to 1$; $A_2 \to$ y;

Jelölés:

Definíció

Relációs séma: $R(A_1, \ldots, A_n)$, *ahol R a reláció neve, az* A_i*-k pedig az attribútumok nevei.*

Például: Személy(Vezetéknév, Keresztnév, Neme, Végzettsége)

3. Gondolhatunk egy relációra úgy is, mint függvények halmazára:

Definíció

Egy sor = egy függvény: $s : \{attribútumok\} \to \{attr.\ értékkészlete\}$
Egy R reláció ilyen függvények halmaza.

Így tényleg nem számít a sorrend, se a sorok között, se az attribútumok között.
Nincs két azonos sor.
Például:
R_1-ben: 1. sor: $A_1 \to 1$; $A_2 \to$ y;

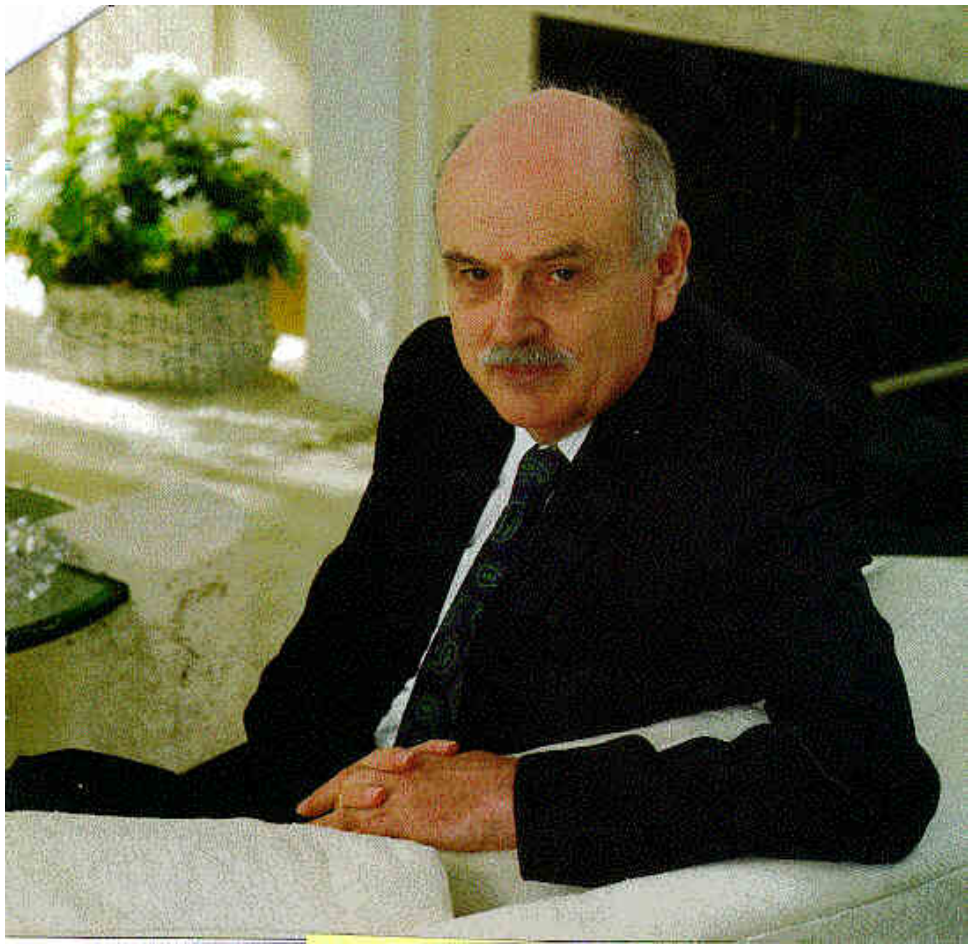
Jelölés:

Definíció

Relációs séma: $R(A_1, \ldots, A_n)$, *ahol R a reláció neve, az* A_i*-k pedig az attribútumok nevei.*

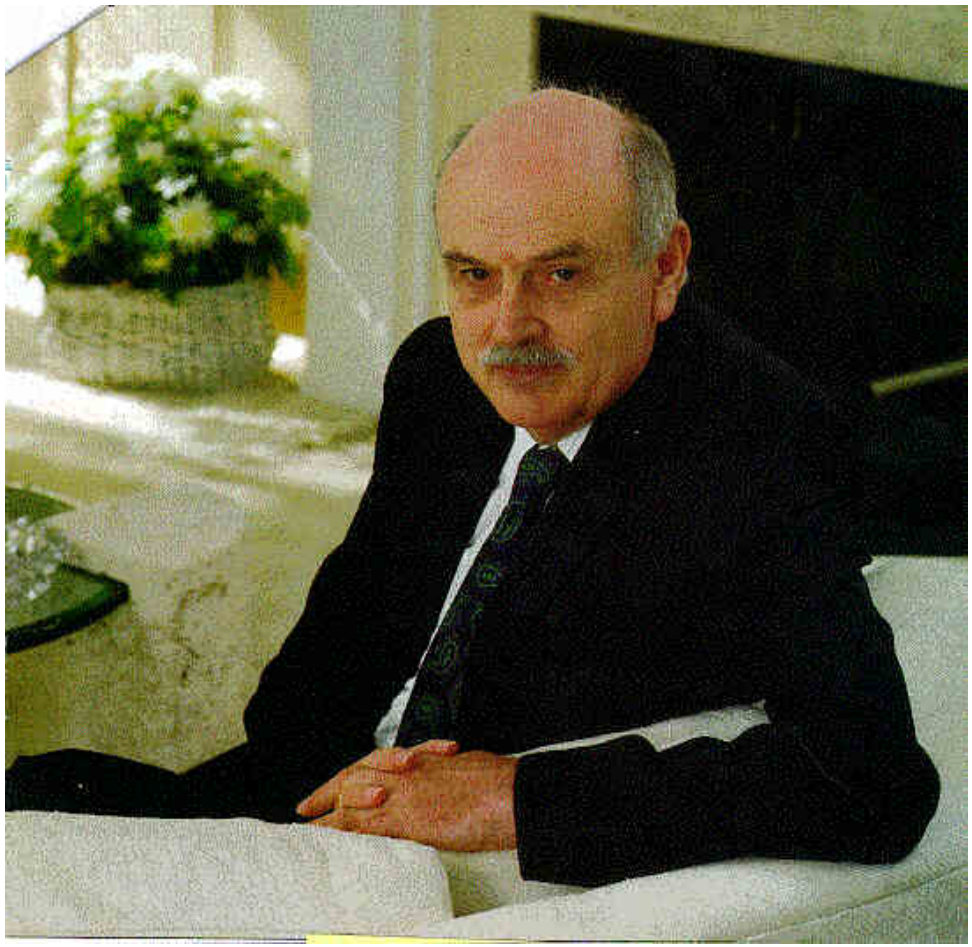
Például: Személy(Vezetéknév, Keresztnév, Neme, Végzettsége)
Gyakorlatban azért mégis rögzítünk egy sorrendet, azt, amelyikben felsoroljuk az attribútumokat.

Relációs modell



Edgar F. Codd, (1932–)
1970-es cikk: *A Relational Model of Data for Large Shared Data Banks*

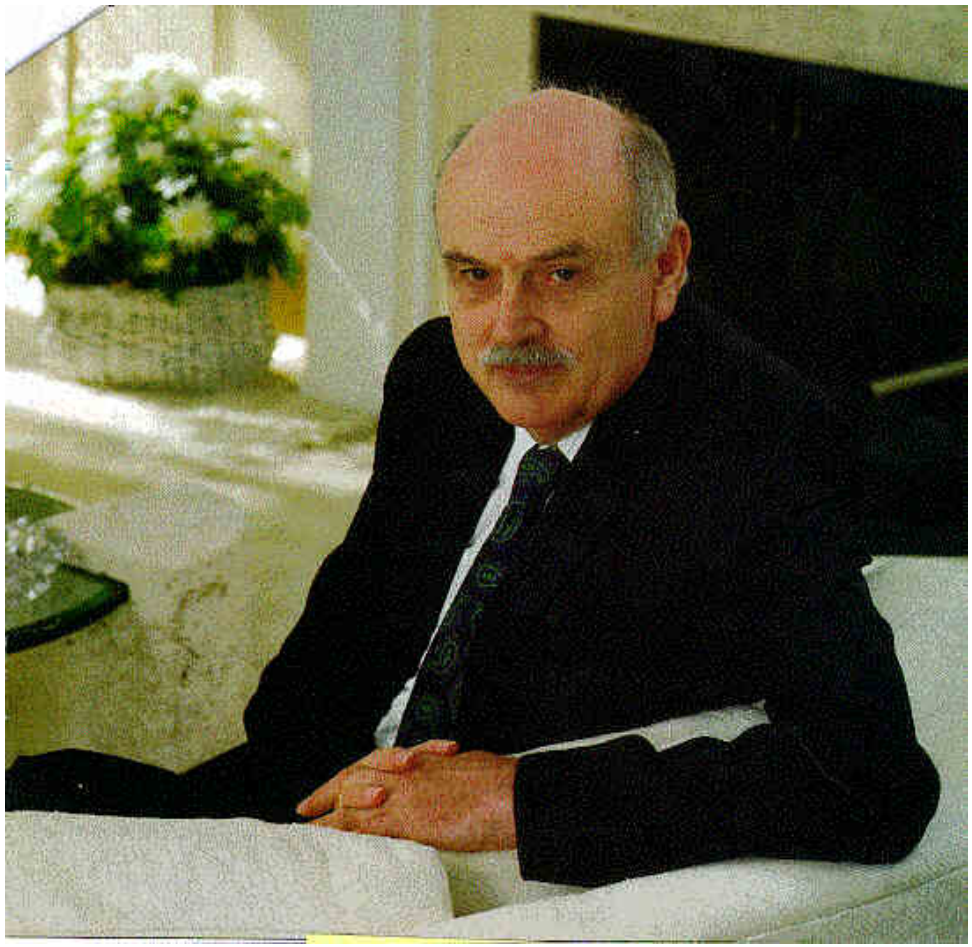
Relációs modell



Edgar F. Codd, (1932–)
1970-es cikk: *A Relational Model of Data for Large Shared Data Banks*

Teljes adatmodell: nem csak azt mondja meg hogyan írok le, hanem vannak műveletek is.

Relációs modell



Edgar F. Codd, (1932–)
1970-es cikk: *A Relational Model of Data for Large Shared Data Banks*

Teljes adatmodell: nem csak azt mondja meg hogyan írok le, hanem vannak műveletek is.
Ezeket a műveleteket relációkra alkalmazhatom és így újabb relációkat kapok majd.

A relációs algebra alapműveletei

- Halmazműveletek (bármilyen halmazra mennének)
  - unió: $\cup$
  - különbség: $\setminus$
  - szorzat: $\times$

A relációs algebra alapműveletei

- Halmazműveletek (bármilyen halmazra mennének)
  - unió: $\cup$
  - különbség: $\setminus$
  - szorzat: $\times$
- Relációs műveletek (ezek már kihasználják, hogy itt relációkról van szó)
  - vetítés, projekció: π
  - kiválasztás, szelekció: σ

A relációs algebra alapműveletei

- Halmazműveletek (bármilyen halmazra mennének)
  - unió: $\cup$
  - különbség: $\setminus$
  - szorzat: $\times$
- Relációs műveletek (ezek már kihasználják, hogy itt relációkról van szó)
  - vetítés, projekció: π
  - kiválasztás, szelekció: σ

Ezek mind tiszta műveletek: reláció $\rightarrow$ reláció

A relációs algebra alapműveletei

- Halmazműveletek (bármilyen halmazra mennének)
  - unió: $\cup$
  - különbség: $\setminus$
  - szorzat: $\times$
- Relációs műveletek (ezek már kihasználják, hogy itt relációkról van szó)
  - vetítés, projekció: π
  - kiválasztás, szelekció: σ

Ezek mind tiszta műveletek: reláció $\rightarrow$ reláció

$\Longrightarrow$ gond nélkül egymásba ágyazhatók

Unió

- R, S relációk $\Longrightarrow R \cup S =$ sorai vagy R vagy S sorai

Unió

- R, S relációk $\Longrightarrow R \cup S =$ sorai vagy R vagy S sorai
  Azonos sorok csak egyszer szerepeljenek.

Unió

- R, S relációk $\Longrightarrow R \cup S =$ sorai vagy R vagy S sorai
  Azonos sorok csak egyszer szerepeljenek. (Gyakorlatban néha lehetnek azonos sorok.)

Unió

- R, S relációk $\Longrightarrow R \cup S =$ sorai vagy R vagy S sorai
  Azonos sorok csak egyszer szerepeljenek. (Gyakorlatban néha lehetnek azonos sorok.)
- csak akkor alkalmazható, ha R és S oszlopszáma egyenlő

Unió

- R, S relációk $\Longrightarrow R \cup S =$ sorai vagy R vagy S sorai
  Azonos sorok csak egyszer szerepeljenek. (Gyakorlatban néha lehetnek azonos sorok.)
- csak akkor alkalmazható, ha R és S oszlopszáma egyenlő
- nem feltétlenül örököl típusokat vagy attribútum neveket

Unió

- R, S relációk $\Longrightarrow R \cup S =$ sorai vagy R vagy S sorai
  Azonos sorok csak egyszer szerepeljenek. (Gyakorlatban néha lehetnek azonos sorok.)
- csak akkor alkalmazható, ha R és S oszlopszáma egyenlő
- nem feltétlenül örököl típusokat vagy attribútum neveket
- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

Unió

- R, S relációk $\Longrightarrow R \cup S =$ sorai vagy R vagy S sorai
  Azonos sorok csak egyszer szerepeljenek. (Gyakorlatban néha lehetnek azonos sorok.)
- csak akkor alkalmazható, ha R és S oszlopszáma egyenlő
- nem feltétlenül örököl típusokat vagy attribútum neveket
- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

$R \cup S$	A	$(R \cup S)_2$
	a	a
	a	c
	b	a
	a	d
	b	b

Különbség

- R, S relációk $\Longrightarrow R \setminus S = R$ azon sorai, amelyek S-ben nem szerepelnek

Különbség

- R, S relációk $\Longrightarrow R \setminus S = R$ azon sorai, amelyek S-ben nem szerepelnek
- nincs kompatibilitási követelmény (Ha pl. különböző az oszlopszám, nem szerepelhetnek azonos sorok úgysem. Ekkor $R \setminus S = R$)

Különbség

- R, S relációk $\Longrightarrow R \setminus S = R$ azon sorai, amelyek S-ben nem szerepelnek
- nincs kompatibilitási követelmény (Ha pl. különböző az oszlopszám, nem szerepelhetnek azonos sorok úgysem. Ekkor $R \setminus S = R$)
- Az eredmény örökli R típusait és attribútum neveit (mert $R \setminus S \subseteq R$)

Különbség

- R, S relációk $\Longrightarrow R \setminus S = R$ azon sorai, amelyek S-ben nem szerepelnek
- nincs kompatibilitási követelmény (Ha pl. különböző az oszlopszám, nem szerepelhetnek azonos sorok úgysem. Ekkor $R \setminus S = R$)
- Az eredmény örökli R típusait és attribútum neveit (mert $R \setminus S \subseteq R$)
- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

Különbség

- R, S relációk $\Longrightarrow R \setminus S = R$ azon sorai, amelyek S-ben nem szerepelnek
- nincs kompatibilitási követelmény (Ha pl. különböző az oszlopszám, nem szerepelhetnek azonos sorok úgysem. Ekkor $R \setminus S = R$)
- Az eredmény örökli R típusait és attribútum neveit (mert $R \setminus S \subseteq R$)
- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

$R \setminus S$	A	B
	b	a

Szorzat (direkt szorzat, Descartes szorzat)

- $R(A_1, \ldots, A_k), S(B_1, \ldots, B_l)$ k ill. l attribútumos relációk $\Longrightarrow R \times S =$ egy $k + l$ attribútumos reláció, R minden sora mögé odatesszük S minden sorát, minden lehetséges módon.

Szorzat (direkt szorzat, Descartes szorzat)

- $R(A_1, \ldots, A_k), S(B_1, \ldots, B_l)$ k ill. l attribútumos relációk $\Longrightarrow R \times S =$ egy $k + l$ attribútumos reláció, R minden sora mögé odatesszük S minden sorát, minden lehetséges módon.
  Ha R-nek n sora van S-nek m sora $\Longrightarrow R \times S$-nek nm sora van

Szorzat (direkt szorzat, Descartes szorzat)

- $R(A_1, \ldots, A_k), S(B_1, \ldots, B_l)$ k ill. l attribútumos relációk $\Longrightarrow R \times S =$ egy $k + l$ attribútumos reláció, R minden sora mögé odatesszük S minden sorát, minden lehetséges módon.
  Ha R-nek n sora van S-nek m sora $\Longrightarrow R \times S$-nek nm sora van
- nincs kompatibilitási követelmény

Szorzat (direkt szorzat, Descartes szorzat)

- $R(A_1, \ldots, A_k), S(B_1, \ldots, B_l)$ k ill. l attribútumos relációk $\Longrightarrow R \times S =$ egy $k + l$ attribútumos reláció, R minden sora mögé odatesszük S minden sorát, minden lehetséges módon.
  Ha R-nek n sora van S-nek m sora $\Longrightarrow R \times S$-nek nm sora van
- nincs kompatibilitási követelmény
- Az eredmény lényegében örökli R és S típusait és attribútum neveit, esetleg át kell nevezni.

Műveletek

- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

Műveletek

- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

$R \times S$	A	B	A'	C
	a	a	a	a
	a	a	a	d
	a	a	a	c
	a	a	b	b

Műveletek

- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

$R \times S$	A	B	A'	C
	a	a	a	a
	a	a	a	d
	a	a	a	c
	a	a	b	b
	a	c	a	a
	a	c	a	d
	a	c	a	c
	a	c	b	b

Műveletek

- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

$R \times S$	A	B	A'	C
	a	a	a	a
	a	a	a	d
	a	a	a	c
	a	a	b	b
	a	c	a	a
	a	c	a	d
	a	c	a	c
	a	c	b	b
	b	a	a	a
	b	a	a	d
	b	a	a	c
	b	a	b	b

Műveletek

- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

$R \times S$	A	B	A'	C
	a	a	a	a
	a	a	a	d
	a	a	a	c
	a	a	b	b
	a	c	a	a
	a	c	a	d
	a	c	a	c
	a	c	b	b
	b	a	a	a
	b	a	a	d
	b	a	a	c
	b	a	b	b

Az unió és különbség könnyű művelet, a szorzat nehezebb. Vigyázni kell mennyit használjuk.

Műveletek

Vetítés

- $R(A_1, \ldots, A_l)$ alakú reláció $\Longrightarrow \pi_{A_{i_1},\ldots,A_{i_n}}(R)$
  R vetítése $A_{i_1}, \ldots, A_{i_n}$-re (fontos a sorrend)

Vetítés

- $R(A_1, \ldots, A_l)$ alakú reláció $\Longrightarrow \pi_{A_{i_1},\ldots,A_{i_n}}(R)$
  R vetítése $A_{i_1}, \ldots, A_{i_n}$-re (fontos a sorrend) $\Longrightarrow$
  Veszem az oszlopokat ebben a sorrendben, a többit eldobom és a többszörös sorokat is eldobom.

Vetítés

- $R(A_1, \ldots, A_l)$ alakú reláció $\Longrightarrow \pi_{A_{i_1},\ldots,A_{i_n}}(R)$
  R vetítése $A_{i_1}, \ldots, A_{i_n}$-re (fontos a sorrend) $\Longrightarrow$
  Veszem az oszlopokat ebben a sorrendben, a többit eldobom és a többszörös sorokat is eldobom.
  Egy oszlop akár többször is szerepelhet. $\Longrightarrow$ átnevezés
- nincs kompatibilitási követelmény (persze amire vetítünk az R-nek attribútuma kell, hogy legyen)

Vetítés

- $R(A_1, \ldots, A_l)$ alakú reláció $\Longrightarrow \pi_{A_{i_1},\ldots,A_{i_n}}(R)$
  R vetítése $A_{i_1}, \ldots, A_{i_n}$-re (fontos a sorrend) $\Longrightarrow$
  Veszem az oszlopokat ebben a sorrendben, a többit eldobom és a többszörös sorokat is eldobom.
  Egy oszlop akár többször is szerepelhet. $\Longrightarrow$ átnevezés
- nincs kompatibilitási követelmény (persze amire vetítünk az R-nek attribútuma kell, hogy legyen)
- Az eredmény örökli R típusait és attribútum neveit

Műveletek

Vetítés

- $R(A_1, \ldots, A_l)$ alakú reláció $\Longrightarrow \pi_{A_{i_1},\ldots,A_{i_n}}(R)$
  R vetítése $A_{i_1}, \ldots, A_{i_n}$-re (fontos a sorrend) $\Longrightarrow$
  Veszem az oszlopokat ebben a sorrendben, a többit eldobom és a többszörös sorokat is eldobom.
  Egy oszlop akár többször is szerepelhet. $\Longrightarrow$ átnevezés
- nincs kompatibilitási követelmény (persze amire vetítünk az R-nek attribútuma kell, hogy legyen)
- Az eredmény örökli R típusait és attribútum neveit
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	c	4

Műveletek

Vetítés

- $R(A_1, \ldots, A_l)$ alakú reláció $\Longrightarrow \pi_{A_{i_1},\ldots,A_{i_n}}(R)$
  R vetítése $A_{i_1}, \ldots, A_{i_n}$-re (fontos a sorrend) $\Longrightarrow$
  Veszem az oszlopokat ebben a sorrendben, a többit eldobom és a többszörös sorokat is eldobom.
  Egy oszlop akár többször is szerepelhet. $\Longrightarrow$ átnevezés
- nincs kompatibilitási követelmény (persze amire vetítünk az R-nek attribútuma kell, hogy legyen)
- Az eredmény örökli R típusait és attribútum neveit
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	c	4

$\pi_A(R)$	A
	a
	b

Műveletek

Vetítés

- $R(A_1, \ldots, A_l)$ alakú reláció $\Longrightarrow \pi_{A_{i_1},\ldots,A_{i_n}}(R)$
  R vetítése $A_{i_1}, \ldots, A_{i_n}$-re (fontos a sorrend) $\Longrightarrow$
  Veszem az oszlopokat ebben a sorrendben, a többit eldobom és a többszörös sorokat is eldobom.
  Egy oszlop akár többször is szerepelhet. $\Longrightarrow$ átnevezés
- nincs kompatibilitási követelmény (persze amire vetítünk az R-nek attribútuma kell, hogy legyen)
- Az eredmény örökli R típusait és attribútum neveit
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	c	4

$\pi_A(R)$	A
	a
	b

$\pi_{C,B,B}(R)$	C	B	B
	2	b	b
	3	c	c
	4	c	c

Kiválasztás, szelekció

- R egy reláció $\Longrightarrow \sigma_F(R) =$ a reláció azon sorai, amelyekre az F formula teljesül.

Kiválasztás, szelekció

- R egy reláció $\Longrightarrow \sigma_F(R) =$ a reláció azon sorai, amelyekre az F formula teljesül.
- Teljesülni fog, hogy $\sigma_F(R) \subseteq R$

Kiválasztás, szelekció

- R egy reláció $\Longrightarrow \sigma_F(R) =$ a reláció azon sorai, amelyekre az F formula teljesül.
- Teljesülni fog, hogy $\sigma_F(R) \subseteq R$
- Nincs megszorítás, csak hogy F értelmes legyen.

Kiválasztás, szelekció

- R egy reláció $\Longrightarrow \sigma_F(R) =$ a reláció azon sorai, amelyekre az F formula teljesül.
- Teljesülni fog, hogy $\sigma_F(R) \subseteq R$
- Nincs megszorítás, csak hogy F értelmes legyen.
- Az eredmény örökli R típusait és attribútum neveit

Kiválasztás, szelekció

- R egy reláció $\Longrightarrow \sigma_F(R) =$ a reláció azon sorai, amelyekre az F formula teljesül.
- Teljesülni fog, hogy $\sigma_F(R) \subseteq R$
- Nincs megszorítás, csak hogy F értelmes legyen.
- Az eredmény örökli R típusait és attribútum neveit
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	c	4

Kiválasztás, szelekció

- R egy reláció $\Longrightarrow \sigma_F(R) =$ a reláció azon sorai, amelyekre az F formula teljesül.
- Teljesülni fog, hogy $\sigma_F(R) \subseteq R$
- Nincs megszorítás, csak hogy F értelmes legyen.
- Az eredmény örökli R típusait és attribútum neveit
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	c	4

$\sigma_{A \neq B \wedge C > 2}(R)$	A	B	C
	a	c	3
	b	c	4

Az F formula:

Atomok: $A\ \theta\ B$, $A\ \theta\ c$, $c\ \theta\ B$,

ahol A, B attribútumok, c érték (konstans), $\theta \in \{<, >, =, \boxed{\leq, \geq, \neq}\}$

Az F formula:

Atomok: $A\ \theta\ B$, $A\ \theta\ c$, $c\ \theta\ B$,
ahol A, B attribútumok, c érték (konstans), $\theta \in \{<, >, =, \boxed{\leq, \geq, \neq}\}$

Építkezés: $\wedge, \vee, \neg, (,)$ Kvantorok, nincsenek!

Az F formula:

Atomok: $A\ \theta\ B$, $A\ \theta\ c$, $c\ \theta\ B$,
ahol A, B attribútumok, c érték (konstans), $\theta \in \{<, >, =, \boxed{\leq, \geq, \neq}\}$

Építkezés: $\wedge, \vee, \neg, (,)$ Kvantorok, nincsenek!

- Példa:

DOLGOZÓ(NÉV,CÍM,FIZETÉS)

$$\sigma_{\text{CÍM}=\text{'BP., Várna u.'} \wedge \text{FIZETÉS} > \text{'150000'}}(\text{DOLGOZÓ})$$

Relációs algebra

Definíció

Alapreláció: *A bevezetés, tervezés során definiált tábla, ami meg van adva.*

Relációs algebra

Definíció

Alapreláció: *A bevezetés, tervezés során definiált tábla, ami meg van adva.*
A relációs algebra relációi: *amik kifejezhetők az alaprelációkból* $\cup, \setminus, \times, \pi, \sigma$ *segítségével.*

Relációs algebra

Definíció

Alapreláció: A bevezetés, tervezés során definiált tábla, ami meg van adva.
A relációs algebra relációi: amik kifejezhetők az alaprelációkból $\cup, \setminus, \times, \pi, \sigma$ segítségével.
Származtatott reláció: nem alapreláció, de kifejezhető.

Relációs algebra

Definíció

Alapreláció: A bevezetés, tervezés során definiált tábla, *ami meg van adva.*
A relációs algebra relációi: amik kifejezhetők az alaprelációkból $\cup, \setminus, \times, \pi, \sigma$ segítségével.
Származtatott reláció: nem alapreláció, de kifejezhető.

Definíció

Egy lekérdező nyelv (igazi vagy modell) relációsan teljes, ha benne megvalósíthatók a relációs algebra alapműveletei: $\cup, \setminus, \times, \pi, \sigma$

Relációs algebra

Definíció

Alapreláció: A bevezetés, tervezés során definiált tábla, *ami meg van adva.*
A relációs algebra relációi: amik kifejezhetők az alaprelációkból $\cup, \setminus, \times, \pi, \sigma$ segítségével.
Származtatott reláció: nem alapreláció, de kifejezhető.

Definíció

Egy lekérdező nyelv (igazi vagy modell) relációsan teljes, ha benne megvalósíthatók a relációs algebra alapműveletei: $\cup, \setminus, \times, \pi, \sigma$

Ez fontos követelmény, általában tudja is mindegyik.

Relációs algebra

Definíció

Alapreláció: A bevezetés, tervezés során definiált tábla, ami meg van adva.
A relációs algebra relációi: amik kifejezhetők az alaprelációkból $\cup, \setminus, \times, \pi, \sigma$ segítségével.
Származtatott reláció: nem alapreláció, de kifejezhető.

Definíció

Egy lekérdező nyelv (igazi vagy modell) relációsan teljes, ha benne megvalósíthatók a relációs algebra alapműveletei: $\cup, \setminus, \times, \pi, \sigma$

Ez fontos követelmény, általában tudja is mindegyik.
Inkább az a baj, hogy néha túl sokat tudnak, de nincs hatékony implementáció.

Adatbázisok elmélete

Származtatott műveletek

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

6. előadás

Származtatott műveletek

Hasznosak, de mivel kifejezhetők az 5 alapművelettel, ezért lényegében csak rövidítések.

Metszet

- R, S relációk $\Longrightarrow R \cap S = R \setminus (R \setminus S)$ azok a sorok, amelyek mindkettőben benne vannak.

Származtatott műveletek

Hasznosak, de mivel kifejezhetők az 5 alapművelettel, ezért lényegében csak rövidítések.

Metszet

- R, S relációk $\Longrightarrow R \cap S = R \setminus (R \setminus S)$ azok a sorok, amelyek mindkettőben benne vannak.
- nincs kompatibilitási követelmény $\Longleftarrow \setminus$ tulajdonságából

Származtatott műveletek

Hasznosak, de mivel kifejezhetők az 5 alapművelettel, ezért lényegében csak rövidítések.

Metszet

- R, S relációk $\Longrightarrow R \cap S = R \setminus (R \setminus S)$ azok a sorok, amelyek mindkettőben benne vannak.
- nincs kompatibilitási követelmény $\Longleftarrow$ \ tulajdonságából
- Az eredmény örökli R típusait és attribútum neveit $\Longleftarrow$ \ tulajdonságából
- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

Származtatott műveletek

Hasznosak, de mivel kifejezhetők az 5 alapművelettel, ezért lényegében csak rövidítések.

Metszet

- R, S relációk $\Longrightarrow R \cap S = R \setminus (R \setminus S)$ azok a sorok, amelyek mindkettőben benne vannak.
- nincs kompatibilitási követelmény $\Longleftarrow$ \ tulajdonságából
- Az eredmény örökli R típusait és attribútum neveit $\Longleftarrow$ \ tulajdonságából
- Példa:

R	A	B
	a	a
	a	c
	b	a

S	A	C
	a	a
	a	d
	a	c
	b	b

$R \cap S$	A	$B \cap C$
	a	a
	a	c

Származtatott műveletek

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t

Származtatott műveletek

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t
  - Vesszük azokat a sorokat, ahol $R.A_1 = S.A_1, \ldots, R.A_k = S.A_k$, a többit kidobjuk.

Származtatott műveletek

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t
  - Vesszük azokat a sorokat, ahol $R.A_1 = S.A_1, \ldots, R.A_k = S.A_k$, a többit kidobjuk.
  - $\forall A_i$-ból az egyik példányt eldobjuk, azaz vetítünk
    $R.A_1, \ldots, R.A_k, R.B_1, \ldots, R.B_r, S.C_1, \ldots, S.C_s$-re

Származtatott műveletek

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t
  - Vesszük azokat a sorokat, ahol $R.A_1 = S.A_1, \ldots, R.A_k = S.A_k$, a többit kidobjuk.
  - $\forall A_i$-ból az egyik példányt eldobjuk, azaz vetítünk $R.A_1, \ldots, R.A_k, R.B_1, \ldots, R.B_r, S.C_1, \ldots, S.C_s$-re
  - Azonos sorokból csak egyet tartunk meg, a többit kidobjuk.

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t
  - Vesszük azokat a sorokat, ahol $R.A_1 = S.A_1, \ldots, R.A_k = S.A_k$, a többit kidobjuk.
  - $\forall A_i$-ból az egyik példányt eldobjuk, azaz vetítünk $R.A_1, \ldots, R.A_k, R.B_1, \ldots, R.B_r, S.C_1, \ldots, S.C_s$-re
  - Azonos sorokból csak egyet tartunk meg, a többit kidobjuk.
- $$R \bowtie S = \pi_{R.A_1,\ldots}(\sigma_{R.A_1=S.A_1,\ldots}(R \times S))$$

$R \bowtie S$-nek $k + r + s$ oszlopa lesz.

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t
  - Vesszük azokat a sorokat, ahol $R.A_1 = S.A_1, \ldots, R.A_k = S.A_k$, a többit kidobjuk.
  - $\forall A_i$-ból az egyik példányt eldobjuk, azaz vetítünk $R.A_1, \ldots, R.A_k, R.B_1, \ldots, R.B_r, S.C_1, \ldots, S.C_s$-re
  - Azonos sorokból csak egyet tartunk meg, a többit kidobjuk.
- $$R \bowtie S = \pi_{R.A_1,\ldots}(\sigma_{R.A_1=S.A_1,\ldots}(R \times S))$$

$R \bowtie S$-nek $k + r + s$ oszlopa lesz.
Ha nincs közös attribútum. $\Longrightarrow R \bowtie S = R \times S$.

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t
  - Vesszük azokat a sorokat, ahol $R.A_1 = S.A_1, \ldots, R.A_k = S.A_k$, a többit kidobjuk.
  - $\forall A_i$-ból az egyik példányt eldobjuk, azaz vetítünk $R.A_1, \ldots, R.A_k, R.B_1, \ldots, R.B_r, S.C_1, \ldots, S.C_s$-re
  - Azonos sorokból csak egyet tartunk meg, a többit kidobjuk.
- $$R \bowtie S = \pi_{R.A_1,\ldots}(\sigma_{R.A_1=S.A_1,\ldots}(R \times S))$$

  $R \bowtie S$-nek $k + r + s$ oszlopa lesz.
  Ha nincs közös attribútum. $\Longrightarrow R \bowtie S = R \times S$.
- nincs kompatibilitási követelmény

Természetes illesztés (natural join)

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \bowtie S =$
  - Vegyük $R \times S$-t
  - Vesszük azokat a sorokat, ahol $R.A_1 = S.A_1, \ldots, R.A_k = S.A_k$, a többit kidobjuk.
  - $\forall A_i$-ból az egyik példányt eldobjuk, azaz vetítünk $R.A_1, \ldots, R.A_k, R.B_1, \ldots, R.B_r, S.C_1, \ldots, S.C_s$-re
  - Azonos sorokból csak egyet tartunk meg, a többit kidobjuk.
- $$R \bowtie S = \pi_{R.A_1,\ldots}(\sigma_{R.A_1=S.A_1,\ldots}(R \times S))$$

  $R \bowtie S$-nek $k + r + s$ oszlopa lesz.
  Ha nincs közös attribútum. $\Longrightarrow R \bowtie S = R \times S$.
- nincs kompatibilitási követelmény
- Az eredmény örökli R és S típusait és attribútum neveit

Természetes illesztés

- Gyakorlatban ennél hatékonyabban számítjuk ki.
- Az oszlopok sorrendje nem definiált, de általában: R oszlopai, aztán S saját oszlopai.
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2

Természetes illesztés

- Gyakorlatban ennél hatékonyabban számítjuk ki.
- Az oszlopok sorrendje nem definiált, de általában: R oszlopai, aztán S saját oszlopai.
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2

$R \bowtie S$	A	B	C	D
	a	b	2	a
	a	b	2	x
	a	c	3	b

Természetes illesztés

- Gyakorlatban ennél hatékonyabban számítjuk ki.
- Az oszlopok sorrendje nem definiált, de általában: R oszlopai, aztán S saját oszlopai.
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2

$R \bowtie S$	A	B	C	D
	a	b	2	a
	a	b	2	x
	a	c	3	b

Miért „természetes”?

Természetes illesztés

Példa: TERMELŐ(TNÉV,TERMÉK,ÁR,CÍM)

Természetes illesztés

Példa: TERMELŐ(TNÉV,TERMÉK,ÁR,CÍM)

- TNÉV $\rightarrow$ CÍM
- TNÉV, TERMÉK $\rightarrow$ ÁR

Természetes illesztés

Példa: TERMELŐ(TNÉV,TERMÉK,ÁR,CÍM)

- TNÉV $\rightarrow$ CÍM
- TNÉV, TERMÉK $\rightarrow$ ÁR

Gond: TERMELŐ címét minden terméknél tároljuk
$\Longrightarrow$ redundancia + veszélyek : cím mindig kell, minden módosításhoz; könnyen sérülhet a fent megadott függés, ha elírom a címet; akkor is kell a cím, ha csak új árut akarok felvenni)

Természetes illesztés

Példa: TERMELŐ(TNÉV,TERMÉK,ÁR,CÍM)

- TNÉV $\rightarrow$ CÍM
- TNÉV, TERMÉK $\rightarrow$ ÁR

Gond: TERMELŐ címét minden terméknél tároljuk
$\Longrightarrow$ redundancia + veszélyek : cím mindig kell, minden módosításhoz; könnyen sérülhet a fent megadott függés, ha elírom a címet; akkor is kell a cím, ha csak új árut akarok felvenni)
Megoldás: Inkább tároljuk két táblában:

Természetes illesztés

Példa: TERMELŐ(TNÉV,TERMÉK,ÁR,CÍM)

- TNÉV $\rightarrow$ CÍM
- TNÉV, TERMÉK $\rightarrow$ ÁR

Gond: TERMELŐ címét minden terméknél tároljuk
$\Longrightarrow$ redundancia + veszélyek : cím mindig kell, minden módosításhoz; könnyen sérülhet a fent megadott függés, ha elírom a címet; akkor is kell a cím, ha csak új árut akarok felvenni)
Megoldás: Inkább tároljuk két táblában:
$R = \pi_{\text{TNÉV, CÍM}}(\text{TERMELŐ})$ és
$S = \pi_{\text{TNÉV, TERMÉK, ÁR}}(\text{TERMELŐ})$

Természetes illesztés

Példa: TERMELŐ(TNÉV,TERMÉK,ÁR,CÍM)

- TNÉV $\rightarrow$ CÍM
- TNÉV, TERMÉK $\rightarrow$ ÁR

Gond: TERMELŐ címét minden terméknél tároljuk
$\Longrightarrow$ redundancia + veszélyek : cím mindig kell, minden módosításhoz; könnyen sérülhet a fent megadott függés, ha elírom a címet; akkor is kell a cím, ha csak új árut akarok felvenni)
Megoldás: Inkább tároljuk két táblában:
$R = \pi_{\text{TNÉV, CÍM}}(\text{TERMELŐ})$ és
$S = \pi_{\text{TNÉV, TERMÉK, ÁR}}(\text{TERMELŐ})$
$\Longrightarrow$ TERMELŐ=$R \bowtie S$ (ha kell egyben a tábla, vissza lehet állítani)

Természetes illesztés

Jó-e bármilyen felbontás?

Természetes illesztés

Jó-e bármilyen felbontás?

$R' = \pi_{\text{TNÉV, CÍM, ÁR}}(\text{TERMELŐ})$ és

$S' = \pi_{\text{TNÉV, TERMÉK}}(\text{TERMELŐ})$

Természetes illesztés

Jó-e bármilyen felbontás?

$R' = \pi_{\text{TNÉV, CÍM, ÁR}}(\text{TERMELŐ})$ és

$S' = \pi_{\text{TNÉV, TERMÉK}}(\text{TERMELŐ})$

$\Longrightarrow$ minden terméknek ugyanannyi lesz az ára (sok ára lesz)

$\Longrightarrow$ TERMELŐ$\subsetneq R' \bowtie S'$

Természetes illesztés

Jó-e bármilyen felbontás?

$R' = \pi_{\text{TNÉV, CÍM, ÁR}}(\text{TERMELŐ})$ és

$S' = \pi_{\text{TNÉV, TERMÉK}}(\text{TERMELŐ})$

$\Longrightarrow$ minden terméknek ugyanannyi lesz az ára (sok ára lesz)

$\Longrightarrow$ TERMELŐ$\subsetneq R' \bowtie S'$

Az lesz majd a kérdés, hogy mik lesznek a jó felbontások?

Bal (jobb) félillesztés

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \rtimes S =$ R azon sorai, amelyhez vannak passzoló sorok S-ben

Bal (jobb) félillesztés

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r)$, $S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \rtimes S =$ R azon sorai, amelyhez vannak passzoló sorok S-ben
  $R \rtimes S = \pi_R(R \bowtie S)$
- $R \rtimes S \subseteq R$
- $R \ltimes S =$ ugyanez jobbról
-

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2

Származtatott műveletek

Bal (jobb) félillesztés

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \rtimes S =$ R azon sorai, amelyhez vannak passzoló sorok S-ben
  $R \rtimes S = \pi_R(R \bowtie S)$
- $R \rtimes S \subseteq R$
- $R \ltimes S =$ ugyanez jobbról
-

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2

$R \rtimes S$	A	B	C
	a	b	2
	a	c	3

Származtatott műveletek

Bal (jobb) félillesztés

- $R(A_1, \ldots, A_k, B_1, \ldots, B_r), S(A_1, \ldots, A_k, C_1, \ldots, C_s)$ relációk
  $\Longrightarrow R \rtimes S =$ R azon sorai, amelyhez vannak passzoló sorok S-ben
  $R \rtimes S = \pi_R(R \bowtie S)$
- $R \rtimes S \subseteq R$
- $R \ltimes S =$ ugyanez jobbról
-

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2

$R \rtimes S$	A	B	C
	a	b	2
	a	c	3

$R \ltimes S$	D	C
	a	2
	b	3
	x	2

Származtatott műveletek

θ-illesztés

- R, S relációk
  $\Longrightarrow R \underset{R.A_i \theta S.B_j}{\bowtie} S = R \times S$ azon sorai, amelyben az adott oszlopok θ relációban vannak

θ-illesztés

- R, S relációk
  $\Longrightarrow R \underset{R.A_i \theta S.B_j}{\bowtie} S = R \times S$ azon sorai, amelyben az adott oszlopok θ relációban vannak
  $R \underset{R.A_i \theta S.B_j}{\bowtie} S = \sigma_{R.A_i \theta S.B_j}(R \times S)$

θ-illesztés

- R, S relációk
  $\Longrightarrow R \underset{R.A_i \theta S.B_j}{\bowtie} S = R \times S$ azon sorai, amelyben az adott oszlopok θ relációban vannak
  $R \underset{R.A_i \theta S.B_j}{\bowtie} S = \sigma_{R.A_i \theta S.B_j}(R \times S)$
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	E
	a	2
	b	3
	x	2

θ-illesztés

- R, S relációk
  $\Longrightarrow R \underset{R.A_i \theta S.B_j}{\bowtie} S = R \times S$ azon sorai, amelyben az adott oszlopok θ relációban vannak
  $R \underset{R.A_i \theta S.B_j}{\bowtie} S = \sigma_{R.A_i \theta S.B_j}(R \times S)$
- Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	E
	a	2
	b	3
	x	2

$R \underset{C \leq E}{\bowtie} S$

A	B	C	D	E
a	b	2	a	2
a	b	2	b	3
a	b	2	x	2
a	c	3	b	3

Példák relációs algebra alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Példák relációs algebra alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

A 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

Példák relációs algebra alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

A 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

A 2004. jan. 15-i befizetett összeg és bevétel:

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\bowtie\text{BEFIZ}\right)\right)$$

Példák relációs algebra alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

A 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

A 2004. jan. 15-i befizetett összeg és bevétel:

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\bowtie\text{BEFIZ}\right)\right)$$

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\right)\bowtie\text{BEFIZ}\right)=$$

Példák relációs algebra alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

A 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

A 2004. jan. 15-i befizetett összeg és bevétel:

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\bowtie\text{BEFIZ}\right)\right)$$

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\right)\bowtie\text{BEFIZ}\right)=$$

$$=\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\right)\ltimes\text{BEFIZ}$$

Példák relációs algebra alkalmazására

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

Példák relációs algebra alkalmazására

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\substack{\text{DB, ÁRUNÉV,}\\ \text{EGYSÉGÁR}}} \Big(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2004-01-15'}} \Big(\text{MENNYISÉG} \bowtie \text{ÁRU} \Big) \Big)$$

$$\pi_{\substack{\text{DB, ÁRUNÉV,}\\ \text{EGYSÉGÁR}}} \Big(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2004-01-15'}} \Big(\text{MENNYISÉG} \Big) \bowtie \text{ÁRU} \Big)$$

Mely nevű áruk azok, amelyekkel van azonos egységárú másik áru?

Mely nevű áruk azok, amelyekkel van azonos egységárú másik áru?
Az ÁRU reláció két sorát kell összevetni.

Mely nevű áruk azok, amelyekkel van azonos egységárú másik áru?
Az ÁRU reláció két sorát kell összevetni.

Átnevezés

- Technikai segítség, ha pl. két relációban ugyanolyan attribútum név van, és direkt szorzatot akarunk. Nem változtatja meg a reláció sorait, csak az attribútumok és a reláció nevét, ezért nem igazi művelet.
- $R(A_1, \ldots, A_n)$ egy reláció
  $\Longrightarrow \rho_{S(B_1,\ldots,B_n)}(R) =$ sorai megegyeznek R soraival, a reláció neve S, attribútumai rendre $B_1, \ldots, B_n$.
- Ha csak a relációt akarjuk átnevezni: $\rho_S(R)$

Megoldás:

$$\text{ÁRU1} = \rho_{\text{ÁRU1(ÁRUKÓD1, ÁRUNÉV1, EGYSÉGÁR1)}}(\text{ÁRU})$$

$$\text{ÁRU2} = \rho_{\text{ÁRU2(ÁRUKÓD2, ÁRUNÉV2, EGYSÉGÁR2)}}(\text{ÁRU})$$

$$\text{ÁRU3} = \text{ÁRU1} \underset{\text{EGYSÉGÁR1 = EGYSÉGÁR2} \wedge \text{ ÁRUKÓD1} \neq \text{ÁRUKÓD2}}{\bowtie} \text{ÁRU2}$$

$$\text{ÁRU4} = \pi_{\text{ÁRUNÉV1}} \left(\text{ÁRU3} \right)$$

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)
A relációk jelentése:
TERMÉK: az adott nevű gyártó gyártja az adott modellszámú és adott típusú (PC, Laptop vagy nyomtató) terméket
PC: modellszám, sebesség megaHz-ben, memória megabájtban, merevlemez gigabájtban, a CD sebessége (pl. 4x), az ár
Laptop: mint PC-nél, plusz a képernyő mérete hüvelykben
Nyomtató: modellszám, színes-e (i/n), típusa (tintasugaras, lézer, mátrix), ára

A modellszámokról feltesszük, hogy egyediek.

További példák

- Melyek azok a PC modellek, amelynek sebessége legalább 150?

További példák

- Melyek azok a PC modellek, amelynek sebessége legalább 150?

$$\pi_{\text{MODELL}} \left(\sigma_{\text{SEBESSÉG} >= 150} (\text{PC}) \right)$$

További példák

- Melyek azok a PC modellek, amelynek sebessége legalább 150?

$$\pi_{\text{MODELL}}\left(\sigma_{\text{SEBESSÉG} >= 150}\left(\text{PC}\right)\right)$$

- Mely gyártók készítenek legalább egy gigás merevlemezű laptopot?

További példák

- Melyek azok a PC modellek, amelynek sebessége legalább 150?

$$\pi_{\text{MODELL}}\left(\sigma_{\text{SEBESSÉG}>=150}\left(\text{PC}\right)\right)$$

- Mely gyártók készítenek legalább egy gigás merevlemezű laptopot?

$$\pi_{\text{GYÁRTÓ}}\left(\text{TERMÉK} \bowtie \sigma_{\text{MEREVLEMEZ}>=1}\left(\text{LAPTOP}\right)\right)$$

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)

- Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)

- Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

$$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{TÍPUS = 'PC'}} \left(\text{TERMÉK} \right) \bowtie \text{PC} \right) \cup$$
$$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{TíPUS = 'LAPTOP'}} \left(\text{TERMÉK} \right) \bowtie \text{LAPTOP} \right) \cup$$
$$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{TíPUS = 'NYOMTATÓ'}} (\text{TERM.}) \bowtie \text{NYOMT. } \right)$$

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)

- Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

$$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{TÍPUS = 'PC'}} \left(\text{TERMÉK} \right) \bowtie \text{PC} \right) \cup$$
$$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{TíPUS = 'LAPTOP'}} \left(\text{TERMÉK} \right) \bowtie \text{LAPTOP} \right) \cup$$
$$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{TíPUS = 'NYOMTATÓ'}} (\text{TERM.}) \bowtie \text{NYOMT. } \right)$$

- Melyek azok a gyártók, akik laptopot gyártanak, de PC-t nem?

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)

- Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

  $$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{ TÍPUS = 'PC'}} \left(\text{TERMÉK}\right) \bowtie \text{PC}\right) \cup$$
  $$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{ TíPUS = 'LAPTOP'}} \left(\text{TERMÉK}\right) \bowtie \text{LAPTOP}\right) \cup$$
  $$\pi_{\text{MODELL, ÁR}} \left(\sigma_{\text{GYÁRTÓ='B'} \wedge \text{ TíPUS = 'NYOMTATÓ'}} (\text{TERM.}) \bowtie \text{NYOMT. }\right)$$

- Melyek azok a gyártók, akik laptopot gyártanak, de PC-t nem?

  $$\text{TERMÉK1} = \rho_{\text{TERMÉK1}} \left(\pi_{\text{GYÁRTÓ, TÍPUS}} \left(\text{TERMÉK}\right)\right)$$
  $$\pi_{\text{GYÁRTÓ}} \left(\sigma_{\text{TÍPUS='LAPTOP'}} \left(\text{TERMÉK1}\right)\right) \setminus \pi_{\text{GYÁRTÓ}} \left(\sigma_{\text{TÍPUS='PC'}} \left(\text{TERMÉK1}\right)\right)$$

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)

További példák

TERMÉK(GYÁRTÓ, MODELL, TÍPUS)
PC(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, CD, ÁR)
LAPTOP(MODELL, SEBESSÉG, MEMÓRIA, MEREVLEMEZ, KÉPERNYŐ, ÁR)
NYOMTATÓ(MODELL, SZÍNES, TÍPUS, ÁR)

- Melyek azok a gyártók, amelyek gyártanak legalább két, egymástól különböző, legalább 133 Mhz-en működő PC-t vagy Laptopot? (Nincs két azonos modellszám!)

$$R1 = \pi_{\text{MODELL, SEBESSÉG}}(\text{PC}) \cup \pi_{\text{MODELL, SEBESSÉG}}(\text{LAPTOP})$$

$$R2 = \pi_{\text{GYÁRTÓ, MODELL}}\left(\sigma_{\text{SEBESSÉG>=133}}(\text{R1}) \bowtie \text{TERMÉK}\right)$$

$$R3 = \rho_{R3(\text{GYÁRTÓ2, MODELL2})}(R2)$$

$$R4 = R2 \underset{\text{GYÁRTÓ = GYÁRTÓ2} \wedge \text{MODELL <> MODELL2}}{\bowtie} R3$$

$$R5 = \pi_{\text{GYÁRTÓ}}(\text{R4})$$

További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:

PC

További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:

További példák

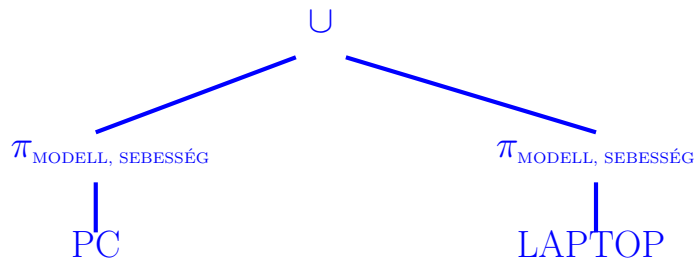
Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:

További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:

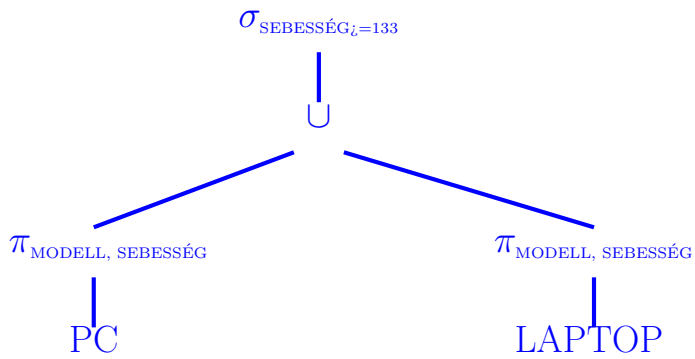
További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



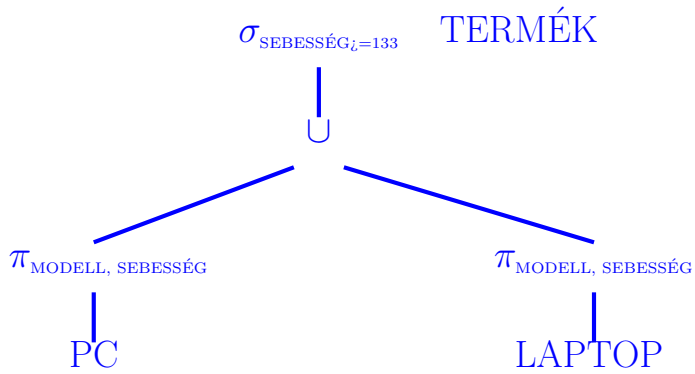
További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



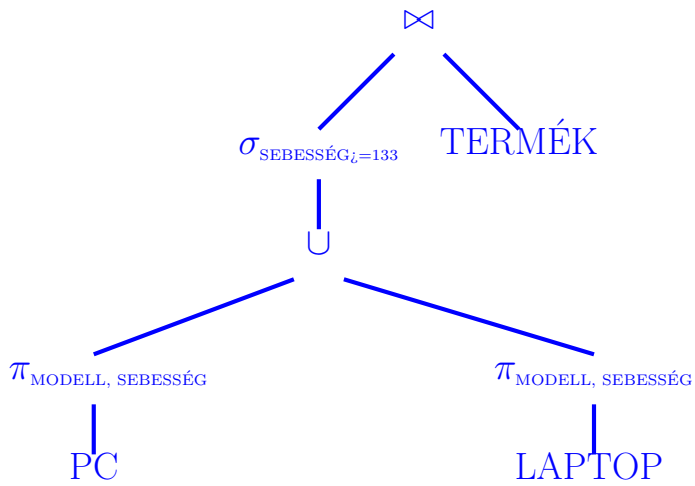
További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



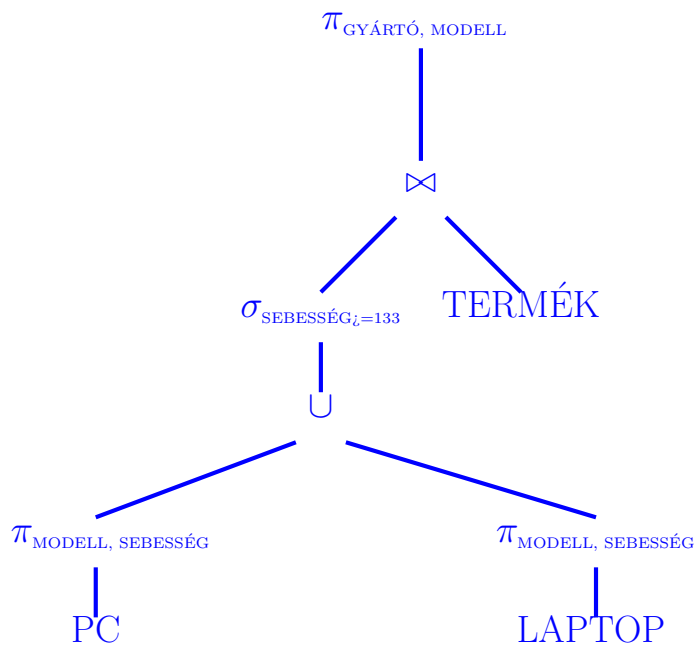
További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



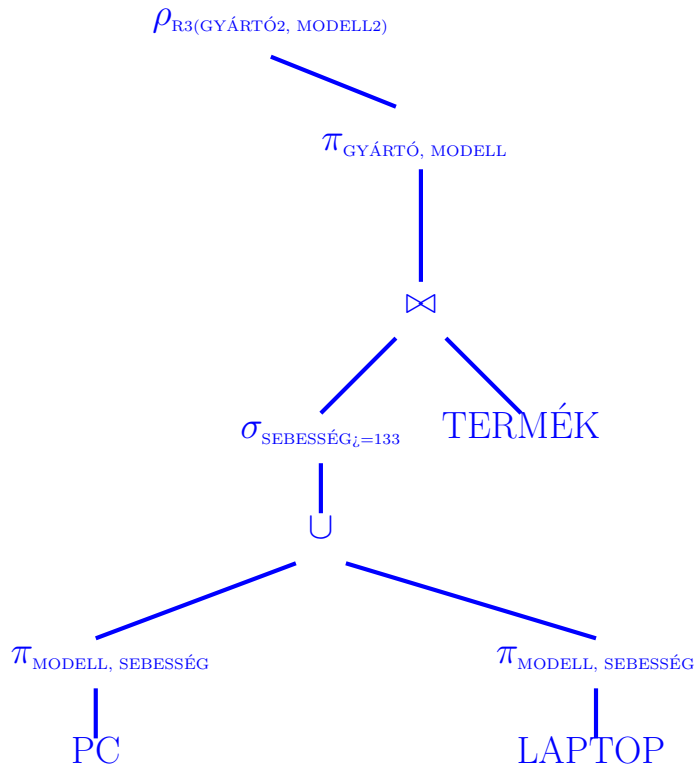
További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



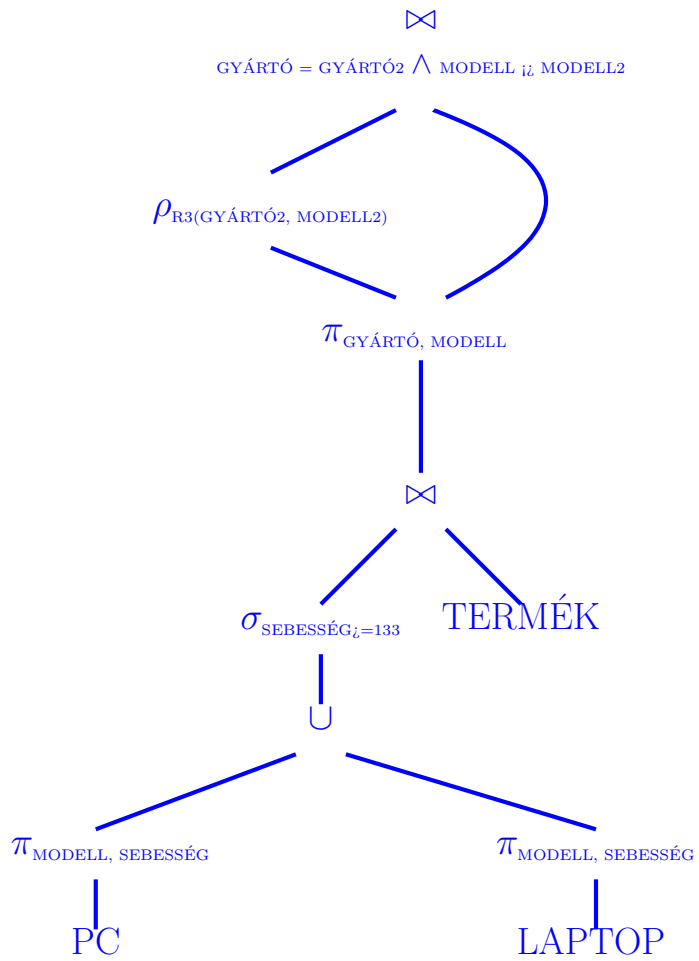
További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



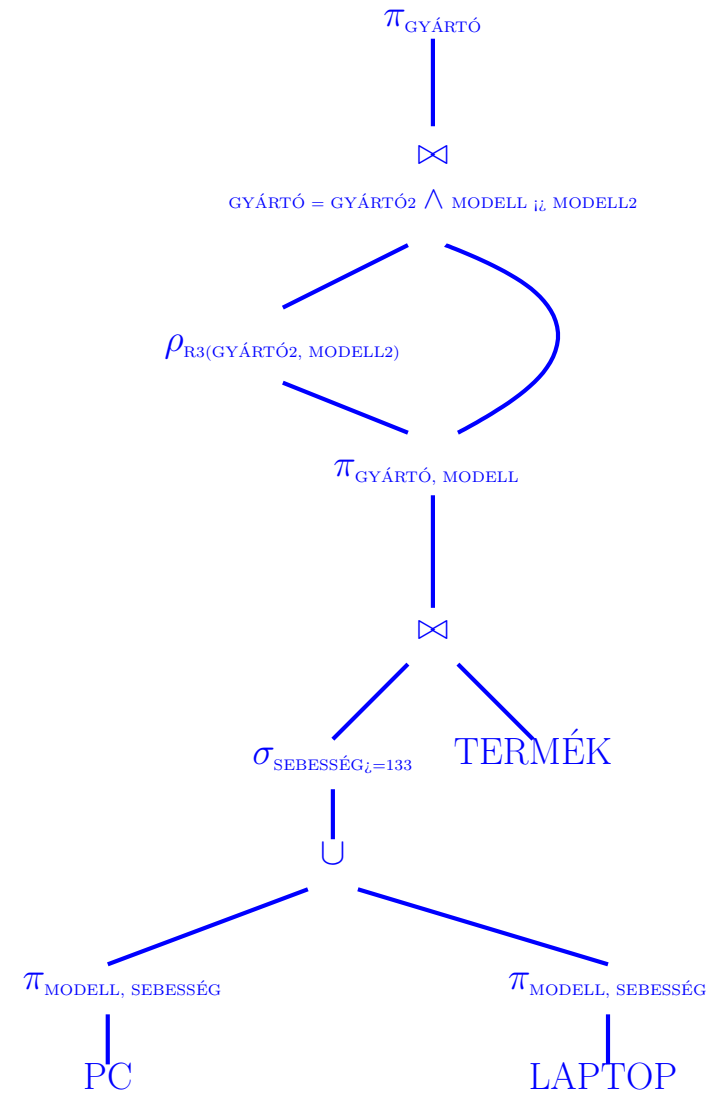
További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



További példák

Megjegyzés: kifejezésfával is meg lehet adni a relációs algebrai kifejezéseket:



További műveletek

Ezek nincsenek benne a relációs algebrában, de fontosak, nem túl műveletigényesek.

aggregátumok: MIN, MAX, AVG, SUM, CNT (szám)

Pl. leggyorsabb gép, átlagár, hányféle printer

eredmény mindig egy szám

További műveletek

Ezek nincsenek benne a relációs algebrában, de fontosak, nem túl műveletigényesek.

aggregátumok: MIN, MAX, AVG, SUM, CNT (szám)
Pl. leggyorsabb gép, átlagár, hányféle printer
eredmény mindig egy szám

aggregátum csoportosítva: Bizonyos feltételek szerinti partíciókban aggregátumok.
Pl. átlagos ár tintasugaras nyomtatók között, egy gyártónak hány terméke van
$\Longrightarrow$ eredmény egy reláció pl. (gyártó, szám) párokból.

További műveletek

rekurzív lezárás: (hagyományos adatkezelésben ritka, intelligensebb rendszerekben inkább)

Pl. reláció: ki főnöke kinek $\Longrightarrow$ lezárás: ki felettese kinek

reláció: melyik városból melyikbe van repülő járat $\Longrightarrow$ lezárás: átszállással el lehet-e jutni

Ezt a relációs algebra nem tudja, csak fix mélységre: pl. max 4 átszállás,

(SQL nem igazán támogatja, de azért kicsit igen: SQL3, majd lesz)

A NULL érték

Lehet, hogy vannak kitöltetlen mezők, ezt meg akarjuk engedni. Pl. ha csak a termelő anyjának neve hiányzik, attól még a termék ára kell.

A NULL érték

Lehet, hogy vannak kitöltetlen mezők, ezt meg akarjuk engedni. Pl. ha csak a termelő anyjának neve hiányzik, attól még a termék ára kell. $\Longrightarrow$ NULL érték

A NULL érték

Lehet, hogy vannak kitöltetlen mezők, ezt meg akarjuk engedni. Pl. ha csak a termelő anyjának neve hiányzik, attól még a termék ára kell. $\Longrightarrow$ NULL érték
2 alapvető értelmezés (majd SQL-nél lesz, hogy hogyan kell megmondani, hogy melyik van éppen, illetve, hogy lehet-e egyáltalán NULL valahol):

- $\nexists$

A NULL érték

Lehet, hogy vannak kitöltetlen mezők, ezt meg akarjuk engedni. Pl. ha csak a termelő anyjának neve hiányzik, attól még a termék ára kell. $\Longrightarrow$ NULL érték
2 alapvető értelmezés (majd SQL-nél lesz, hogy hogyan kell megmondani, hogy melyik van éppen, illetve, hogy lehet-e egyáltalán NULL valahol):

- $\nexists$
- $\exists$, de nem ismerjük.

A NULL érték

Lehet, hogy vannak kitöltetlen mezők, ezt meg akarjuk engedni. Pl. ha csak a termelő anyjának neve hiányzik, attól még a termék ára kell. $\Longrightarrow$ NULL érték
2 alapvető értelmezés (majd SQL-nél lesz, hogy hogyan kell megmondani, hogy melyik van éppen, illetve, hogy lehet-e egyáltalán NULL valahol):

- $\nexists$
- $\exists$, de nem ismerjük.

Attól függően, hogy hogyan értelmezzük a NULL-t:
Mi legyen egy ilyen kérdéssel?:
Pl. $\sigma_{\text{CÍM='BP'}}\left(\text{TERMELŐ}\right)$

A NULL érték

Lehet, hogy vannak kitöltetlen mezők, ezt meg akarjuk engedni. Pl. ha csak a termelő anyjának neve hiányzik, attól még a termék ára kell. $\Longrightarrow$ NULL érték
2 alapvető értelmezés (majd SQL-nél lesz, hogy hogyan kell megmondani, hogy melyik van éppen, illetve, hogy lehet-e egyáltalán NULL valahol):

- $\nexists$
- $\exists$, de nem ismerjük.

Attól függően, hogy hogyan értelmezzük a NULL-t:
Mi legyen egy ilyen kérdéssel?:
Pl. $\sigma_{\text{CÍM='BP'}}\left(\text{TERMELŐ}\right)$
Ilyenkor belevegyük-e ha a cím NULL?

Külső illesztés (outer join)

Definíció

*R, S relációk $\Longrightarrow$ $R \overset{\leftarrow}{\bowtie} S$ **bal külső illesztés**: $R \bowtie S$-hez azokat az R-beli sorokat is hozzáveszük, amihez nem illeszkedik S-beli. Hiányzó helyekre NULL kerül.*

Külső illesztés (outer join)

Definíció

*R, S relációk $\Longrightarrow$ $R \overset{\text{bal}}{\bowtie} S$ **bal külső illesztés**: $R \bowtie S$-hez azokat az R-beli sorokat is hozzáveszük, amihez nem illeszkedik S-beli. Hiányzó helyekre NULL kerül.*

Pl. SZEMÉLY(NÉV, KÓD), CÍM(KÓD, CÍM)
SZEMÉLY $\bowtie$ CÍM $\Longrightarrow$ akinek nincs címe nem lesz rajta
SZEMÉLY $\overset{\text{bal}}{\bowtie}$ CÍM $\Longrightarrow$ kiderül, kinek nincs meg a címe

Külső illesztés (outer join)

Definíció

*R, S relációk $\Longrightarrow$ $R \overset{\text{bal}}{\bowtie} S$ **bal külső illesztés**: $R \bowtie S$-hez azokat az R-beli sorokat is hozzáveszük, amihez nem illeszkedik S-beli. Hiányzó helyekre NULL kerül.*

Pl. SZEMÉLY(NÉV, KÓD), CÍM(KÓD, CÍM)
SZEMÉLY $\bowtie$ CÍM $\Longrightarrow$ akinek nincs címe nem lesz rajta
SZEMÉLY $\overset{\text{bal}}{\bowtie}$ CÍM $\Longrightarrow$ kiderül, kinek nincs meg a címe
SQL-ben van, relációs algebrával elvileg nem fejezhető ki (NULL miatt), de elkerülhető.

Külső illesztés (outer join)

Definíció

*R, S relációk $\Longrightarrow$ $R \overset{\rightarrow}{\bowtie} S$ **bal külső illesztés**: $R \bowtie S$-hez azokat az R-beli sorokat is hozzáveszük, amihez nem illeszkedik S-beli. Hiányzó helyekre NULL kerül.*

Pl. SZEMÉLY(NÉV, KÓD), CÍM(KÓD, CÍM)
SZEMÉLY $\bowtie$ CÍM $\Longrightarrow$ akinek nincs címe nem lesz rajta
SZEMÉLY $\overset{\rightarrow}{\bowtie}$ CÍM $\Longrightarrow$ kiderül, kinek nincs meg a címe
SQL-ben van, relációs algebrával elvileg nem fejezhető ki (NULL miatt), de elkerülhető.
Ha a relációs algebrát úgy definiáljuk, hogy kiindulhatuk konstans relációból is, akkor:
$(R \bowtie S) \cup (R \setminus (R \overset{\rightarrow}{\bowtie} S)) \times \{(\text{NULL}, \ldots, \text{NULL})\}$

Külső illesztés (outer join)

Definíció

*R, S relációk $\Longrightarrow$ $R \overset{\rightarrow}{\bowtie} S$ **bal külső illesztés**: $R \bowtie S$-hez azokat az R-beli sorokat is hozzáveszük, amihez nem illeszkedik S-beli. Hiányzó helyekre NULL kerül.*

Pl. SZEMÉLY(NÉV, KÓD), CÍM(KÓD, CÍM)
SZEMÉLY $\bowtie$ CÍM $\Longrightarrow$ akinek nincs címe nem lesz rajta
SZEMÉLY $\overset{\rightarrow}{\bowtie}$ CÍM $\Longrightarrow$ kiderül, kinek nincs meg a címe
SQL-ben van, relációs algebrával elvileg nem fejezhető ki (NULL miatt), de elkerülhető.
Ha a relációs algebrát úgy definiáljuk, hogy kiindulhatuk konstans relációból is, akkor:
$(R \bowtie S) \cup (R \setminus (R \overset{\rightarrow}{\bowtie} S)) \times \{(\text{NULL}, \ldots, \text{NULL})\}$
Van **jobb külső illesztés** is: $R \overset{\leftarrow}{\bowtie} S$
Teljes külső illesztés: $R \overset{\leftrightarrow}{\bowtie} S := (R \overset{\rightarrow}{\bowtie} S) \cup (R \overset{\leftarrow}{\bowtie} S)$

Külső illesztés (outer join)

Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2
	y	1

Külső illesztés (outer join)

Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2
	y	1

$R \rtimes S$	A	B	C	D
	a	b	2	a
	a	b	2	x
	a	c	3	b
	b	**a**	**4**	NULL

Külső illesztés (outer join)

Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2
	y	1

Külső illesztés (outer join)

Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2
	y	1

$R \bowtie S$	A	B	C	D
	a	b	2	a
	a	b	2	x
	a	c	3	b
	NULL	NULL	**1**	**y**

Külső illesztés (outer join)

Példa:

R	A	B	C
	a	b	2
	a	c	3
	b	a	4

S	D	C
	a	2
	b	3
	x	2
	y	1

$R \bowtie S$	A	B	C	D
	a	b	2	a
	a	b	2	x
	a	c	3	b
	NULL	NULL	**1**	**y**

$R \bowtie S$	A	B	C	D
	a	b	2	a
	a	b	2	x
	a	c	3	b
	b	**a**	**4**	NULL
	NULL	NULL	**1**	**y**

Külső unió

Részben kompatibilis relációk egyesítésére:
DIÁK(NÉV, TÉMAVEZ, TSZK)
TANÁR(NÉV, TSZK, BEOSZT)

DIÁK $\cup_k$ TANÁR

	NÉV	TSZK	TÉMAVEZ	BEOSZT
diák				NULL
tanár			NULL	

Multihalmazos szemantika

A relációs algebrában ugyan minden reláció halmaz, ezért nincsenek többszörös sorok, de pl. SQL-nél lesznek. A multihalmazokkal kicsit máshogy vannak a halmazműveletek:

Multihalmazos szemantika

A relációs algebrában ugyan minden reláció halmaz, ezért nincsenek többszörös sorok, de pl. SQL-nél lesznek. A multihalmazokkal kicsit máshogy vannak a halmazműveletek:

Ha a t sor $m_R(t)$ példányban van meg R-ben és $m_S(t)$ példányban van meg S-ben, akkor

- $m_{(R \cup S)}(t) := m_R(t) + m_S(t)$ példányban lesz meg R és S uniójában

Multihalmazos szemantika

A relációs algebrában ugyan minden reláció halmaz, ezért nincsenek többszörös sorok, de pl. SQL-nél lesznek. A multihalmazokkal kicsit máshogy vannak a halmazműveletek:

Ha a t sor $m_R(t)$ példányban van meg R-ben és $m_S(t)$ példányban van meg S-ben, akkor

- $m_{(R \cup S)}(t) := m_R(t) + m_S(t)$ példányban lesz meg R és S uniójában
- $m_{(R \cap S)}(t) := \min\{m_R(t), m_S(t)\}$ példányban lesz meg R és S metszetében

Multihalmazos szemantika

A relációs algebrában ugyan minden reláció halmaz, ezért nincsenek többszörös sorok, de pl. SQL-nél lesznek. A multihalmazokkal kicsit máshogy vannak a halmazműveletek:

Ha a t sor $m_R(t)$ példányban van meg R-ben és $m_S(t)$ példányban van meg S-ben, akkor

- $m_{(R \cup S)}(t) := m_R(t) + m_S(t)$ példányban lesz meg R és S uniójában
- $m_{(R \cap S)}(t) := \min\{m_R(t), m_S(t)\}$ példányban lesz meg R és S metszetében
- $m_{(R \setminus S)}(t) := \max\{m_R(t) - m_S(t), 0\})$ példányban lesz meg $R \setminus S$-ben

Multihalmazos szemantika

A relációs algebrában ugyan minden reláció halmaz, ezért nincsenek többszörös sorok, de pl. SQL-nél lesznek. A multihalmazokkal kicsit máshogy vannak a halmazműveletek:

Ha a t sor $m_R(t)$ példányban van meg R-ben és $m_S(t)$ példányban van meg S-ben, akkor

- $m_{(R \cup S)}(t) := m_R(t) + m_S(t)$ példányban lesz meg R és S uniójában
- $m_{(R \cap S)}(t) := \min\{m_R(t), m_S(t)\}$ példányban lesz meg R és S metszetében
- $m_{(R \setminus S)}(t) := \max\{m_R(t) - m_S(t), 0\})$ példányban lesz meg $R \setminus S$-ben
- Kiválasztásnál, vetítésnél nincs változás.

ODL séma átírása relációsémává

Legegyszerűbb eset $\Longrightarrow$ az osztályoknak csak attribútumai vannak, amik atomi típusúak:

```
interface Film (keys (cím, év)){
    attribute cím;
    attribute év;
    attribute hossz;
    attribute szalagFajta;
```

ODL séma átírása relációsémává

Legegyszerűbb eset $\Longrightarrow$ az osztályoknak csak attribútumai vannak, amik atomi típusúak:

```
interface Film (keys (cím, év)){
    attribute cím;
    attribute év;
    attribute hossz;
    attribute szalagFajta;
};
```

$\Longrightarrow$ Film(<u>cím</u>, <u>év</u>, hossz, szalagFajta)

ODL séma átírása relációsémává

Legegyszerűbb eset $\Longrightarrow$ az osztályoknak csak attribútumai vannak, amik atomi típusúak:

```
interface Film (keys (cím, év)){
    attribute cím;
    attribute év;
    attribute hossz;
    attribute szalagFajta;
};
```

$\Longrightarrow$ Film(<u>cím</u>, <u>év</u>, hossz, szalagFajta)

A kulcs az ODL-es kulcs lesz, ha egy van csak belőle; ezt aláhúzással jelöljük. Ha több kulcs van: egyiket jelöljük, a többit írásban (de persze azok is kulcsok a relációsémában is).

ODL séma átírása relációsémává

Legegyszerűbb eset $\Longrightarrow$ az osztályoknak csak attribútumai vannak, amik atomi típusúak:

```
interface Film (keys (cím, év)){
    attribute cím;
    attribute év;
    attribute hossz;
    attribute szalagFajta;
};
```

$\Longrightarrow$ Film(<u>cím</u>, <u>év</u>, hossz, szalagFajta)

A kulcs az ODL-es kulcs lesz, ha egy van csak belőle; ezt aláhúzással jelöljük. Ha több kulcs van: egyiket jelöljük, a többit írásban (de persze azok is kulcsok a relációsémában is).

ODL séma átírása relációsémává

Összetett típusú attribútumok: pl. rekordszerkezet OK $\Longrightarrow$

```
interface Színész (key név){
    attribute String név;
    attribute Struct Cím
        { string város, string utca } lakcím;
```

ODL séma átírása relációsémává

Összetett típusú attribútumok: pl. rekordszerkezet OK $\Longrightarrow$

```
interface Színész (key név){
    attribute String név;
    attribute Struct Cím
        { string város, string utca } lakcím;
};
```

$\Longrightarrow$ Színész(<u>név</u>, város, utca)

ODL séma átírása relációsémává

Összetett típusú attribútumok: pl. rekordszerkezet OK $\Longrightarrow$

```
interface Színész (key név){
    attribute String név;
    attribute Struct Cím
        { string város, string utca } lakcím;
};
```

$\Longrightarrow$ Színész(<u>név</u>, város, utca)

Kulcs: ugyanúgy, mint a nem összetett attribútumnál

De pl. halmaz csak rosszul oldható meg: halmaz minden eleméhez új sor.

```
interface Színész (key név) {
    attribute String név;
    attribute Set <
        Struct Cím{ string város, string utca }
        > lakcím;
};
```

De pl. halmaz csak rosszul oldható meg: halmaz minden eleméhez új sor.

```
interface Színész (key név) {
    attribute String név;
    attribute Set <
        Struct Cím{ string város, string utca }
        > lakcím;
};
```

$\Longrightarrow$ Színész(név, város, utca)

De pl. halmaz csak rosszul oldható meg: halmaz minden eleméhez új sor.

```
interface Színész (key név) {
    attribute String név;
    attribute Set <
        Struct Cím{ string város, string utca }
        > lakcím;
};
```

$\Longrightarrow$ `Színész(név, város, utca)`

név	város	utca
Gálvölgyi J.	Budapest	Nyereg u. 2.
Gálvölgyi J.	Budapest	Kantár u. 3.
⋮		

Kulcs: elromlik az ODL-es kulcs, lehet, hogy ami ott kulcs volt, itt már nem lesz az

$\Longrightarrow$ baj

Kapcsolatok átírása

Ha valamelyik irányba egyértékű a kapcsolat: ha a C és D közti kapcsolat D felé egyirányú, akkor a C osztály átírásakor bevesszük a D osztály kulcsát is (ha több van, akkor egyet)

Kapcsolatok átírása

Ha valamelyik irányba egyértékű a kapcsolat: ha a *C* és *D* közti kapcsolat *D* felé egyirányú, akkor a *C* osztály átírásakor bevesszük a *D* osztály kulcsát is (ha több van, akkor egyet)
Csak egyik irányból csináljuk, akkor is, ha a másik irányba is „egy” a kapcsolat

Kapcsolatok átírása

Ha valamelyik irányba egyértékű a kapcsolat: ha a *C* és *D* közti kapcsolat *D* felé egyirányú, akkor a *C* osztály átírásakor bevesszük a *D* osztály kulcsát is (ha több van, akkor egyet)
Csak egyik irányból csináljuk, akkor is, ha a másik irányba is „egy” a kapcsolat

```
interface Film (keys (cím, év)){
    attribute cím;
    attribute év;
    attribute hossz;
    attribute szalagFajta;
    relationship Stúdió gyártó
        inverse Stúdió::gyárt;
```

Kapcsolatok átírása

Ha valamelyik irányba egyértékű a kapcsolat: ha a *C* és *D* közti kapcsolat *D* felé egyirányú, akkor a *C* osztály átírásakor bevesszük a *D* osztály kulcsát is (ha több van, akkor egyet)
Csak egyik irányból csináljuk, akkor is, ha a másik irányba is „egy” a kapcsolat

```
interface Film (keys (cím, év)){
    attribute cím;
    attribute év;
    attribute hossz;
    attribute szalagFajta;
    relationship Stúdió gyártó
        inverse Stúdió::gyárt;};
```

$\Longrightarrow$ `Film(cím, év, hossz, szalagFajta,` **`stúdióNév`**`)`
Feltéve, hogy a stúdiónév kulcs a Stúdió osztályban

Kapcsolatok átírása

Ha valamelyik irányba egyértékű a kapcsolat: ha a *C* és *D* közti kapcsolat *D* felé egyirányú, akkor a *C* osztály átírásakor bevesszük a *D* osztály kulcsát is (ha több van, akkor egyet)
Csak egyik irányból csináljuk, akkor is, ha a másik irányba is „egy” a kapcsolat

```
interface Film (keys (cím, év)){
    attribute cím;
    attribute év;
    attribute hossz;
    attribute szalagFajta;
    relationship Stúdió gyártó
        inverse Stúdió::gyárt;};
```

$\Longrightarrow$ `Film(cím, év, hossz, szalagFajta,` **`stúdióNév`**`)`
Feltéve, hogy a stúdiónév kulcs a Stúdió osztályban

Kapcsolatok átírása

Kulcs: mivel a kapcsolat „egy” jellegű volt, ezért az osztály kulcsa jó lesz kulcsnak a relációsémában is

Kapcsolatok átírása

Kulcs: mivel a kapcsolat „egy” jellegű volt, ezért az osztály kulcsa jó lesz kulcsnak a relációsémában is

Mindkét irányban többértékű kapcsolat: Ugyanaz a probléma, mint a halmaz típusú attribútum.

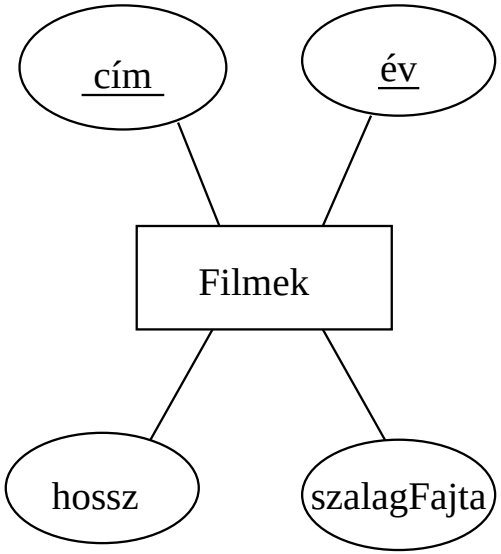
Kapcsolatok átírása

Kulcs: mivel a kapcsolat „egy” jellegű volt, ezért az osztály kulcsa jó lesz kulcsnak a relációsémában is

Mindkét irányban többértékű kapcsolat: Ugyanaz a probléma, mint a halmaz típusú attribútum. Nem lehet jól megoldani, sok sor lesz és a kulcs is elromlik. Ha több ilyen kapcsolat is van $\Longrightarrow$ katasztrófa

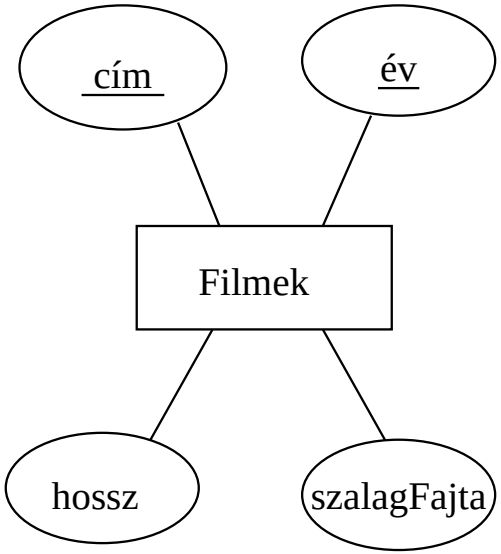
E/K modell átírása

Egyedhalmaz attribútumokkal:



E/K modell átírása

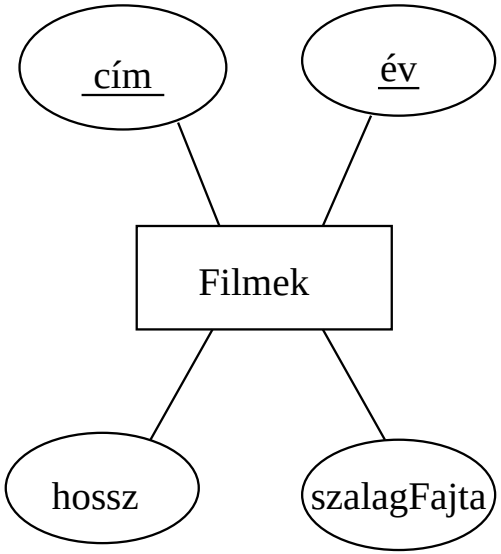
Egyedhalmaz attribútumokkal:



$\Longrightarrow$ `Film(cím, év, hossz, szalagFajta)`

E/K modell átírása

Egyedhalmaz attribútumokkal:



$\Longrightarrow$ `Film(cím, év, hossz, szalagFajta)`

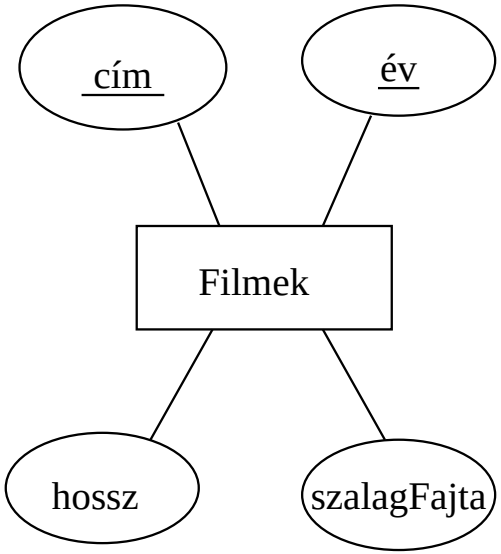
A reláció kulcsa = az egyedhalmaz kulcsa

Egyértékű és többértékű kapcsolat:

$\Longrightarrow$ Külön reláció, attribútumai: a kapcsolatban résztvevő egyedhalmaz kulcsainak uniója + kapcsolat attribútumai (esetleg átnevezés)

E/K modell átírása

Egyedhalmaz attribútumokkal:



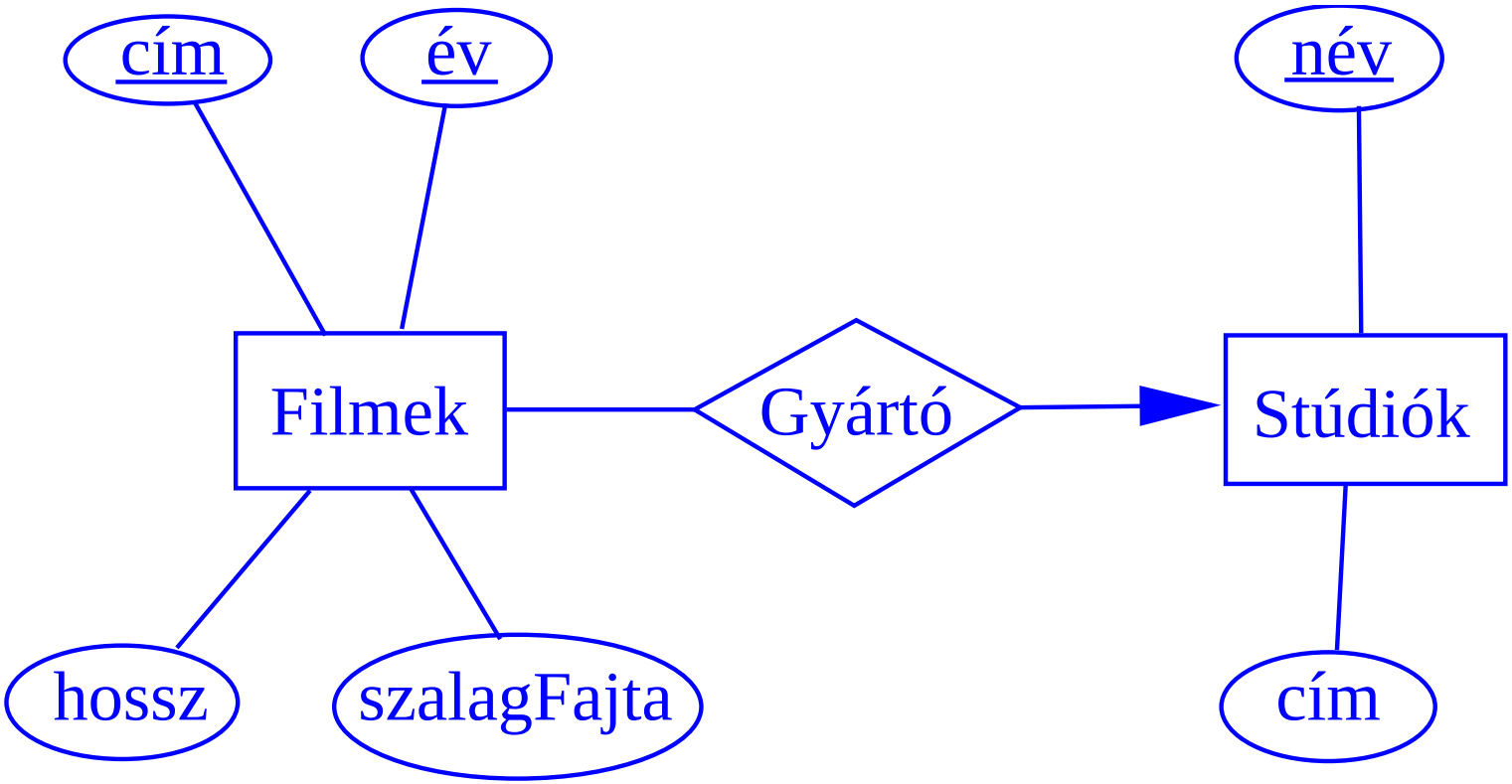
$\Longrightarrow$ `Film(cím, év, hossz, szalagFajta)`

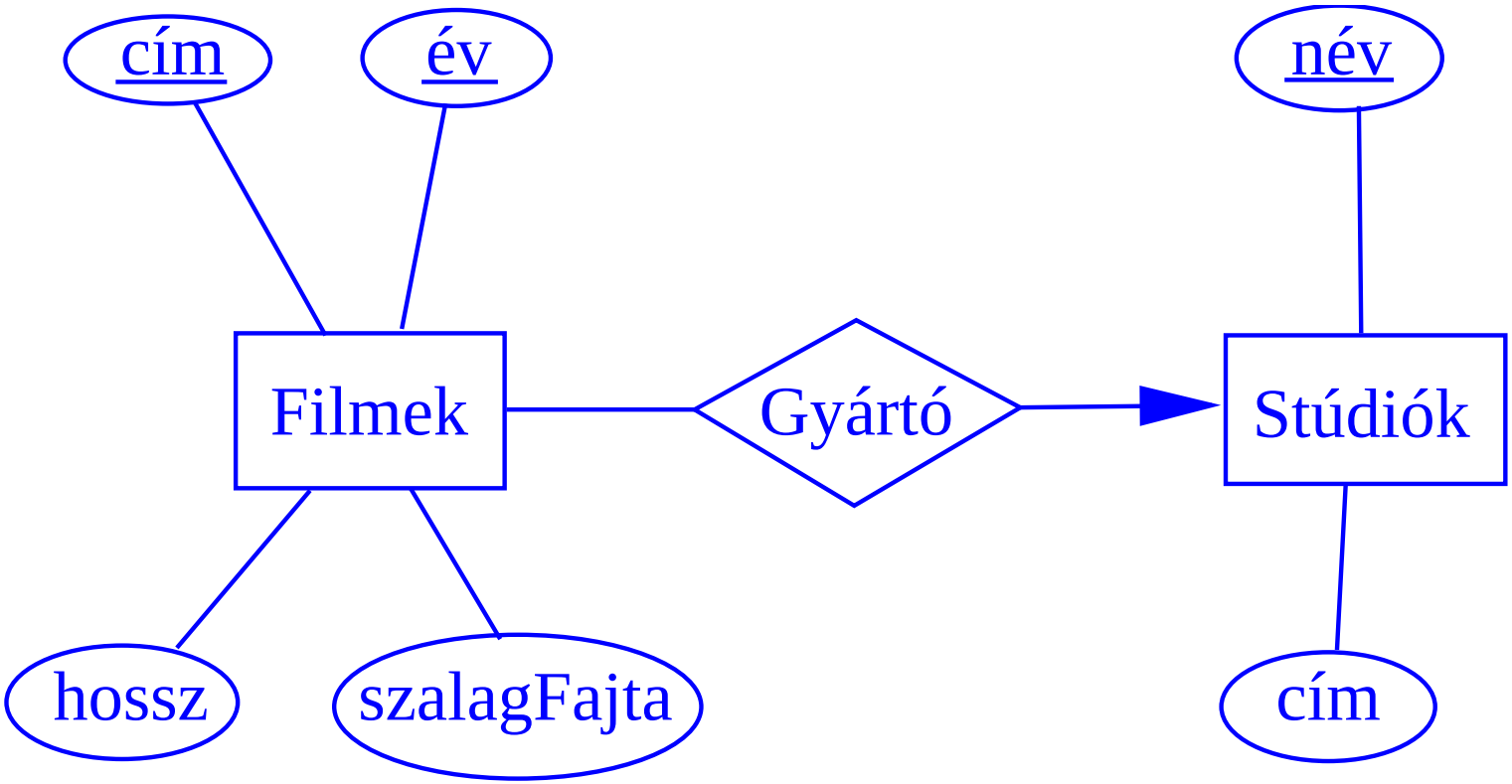
A reláció kulcsa = az egyedhalmaz kulcsa

Egyértékű és többértékű kapcsolat:

$\Longrightarrow$ Külön reláció, attribútumai: a kapcsolatban résztvevő egyedhalmaz kulcsainak uniója + kapcsolat attribútumai (esetleg átnevezés)

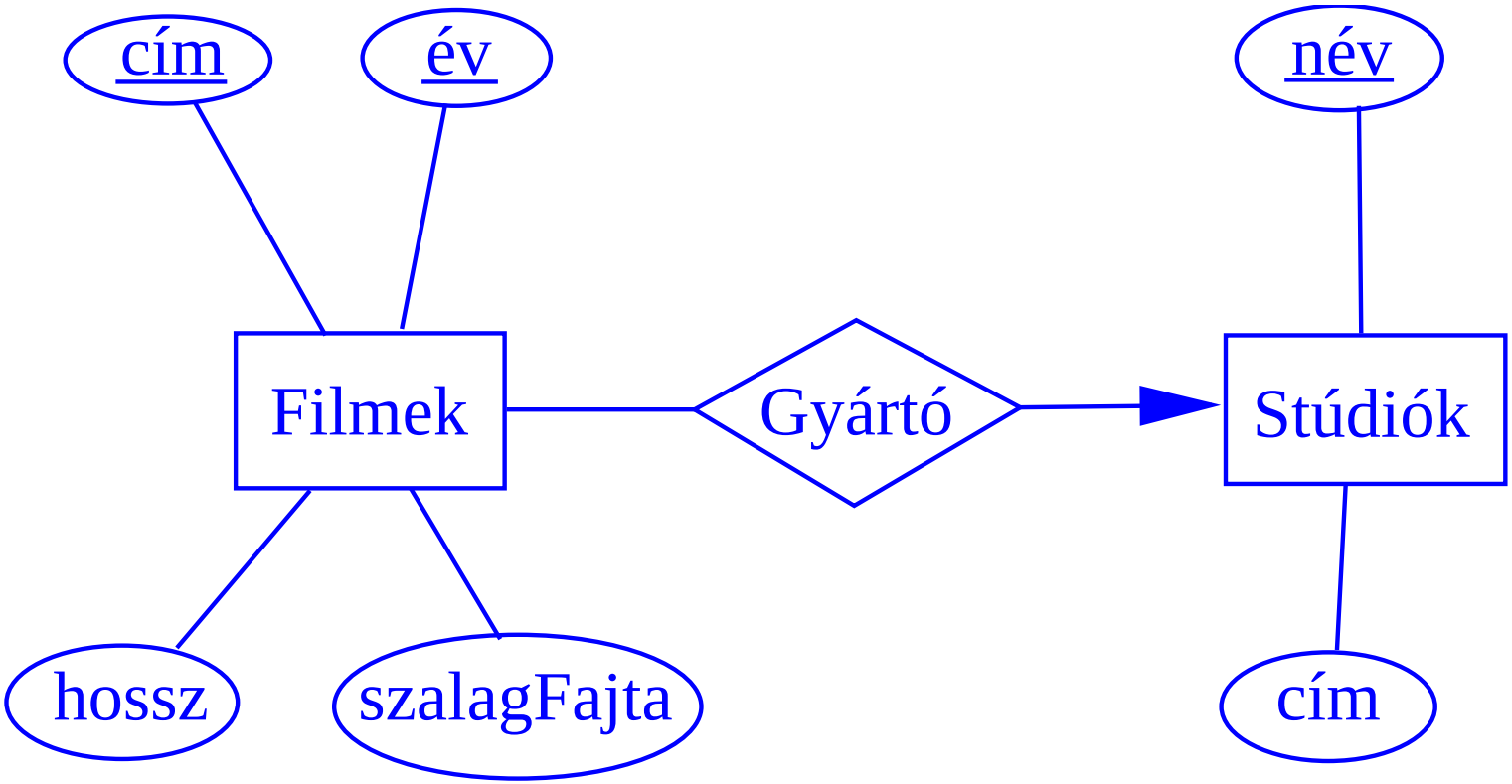
Az így kapott reláció kulcsa: a kapcsolatban résztvevő egyedhalmaz kulcsainak uniója


cím
év
név
Filmek
Gyártó
Stúdiók
hossz
szalagFajta
cím



$\Longrightarrow$

Gyártó(cím, év, stúdióNév)
Film(cím, év, hossz, szalagFajta)
Stúdiók(stúdióNév, cím)



$\Longrightarrow$

Gyártó(cím, év, stúdióNév)
Film(cím, év, hossz, szalagFajta)
Stúdiók(stúdióNév, cím)

E/K-ból jobban lehet relációsat csinálni.

Megjegyzés: ha bináris több-egy kapcsolatról van szó, akkor van jobb megoldás is, az amit ODL-nél csináltunk:
ha az E és F közti kapcsolat F felé egyirányú, akkor az E egyedhalmaz átírásakor bevesszük az F osztály kulcsát is. Ez ugyanazért lesz jó, miért az ODL-es és így eggyel kevesebb tábla lesz.

Megjegyzés: ha bináris több-egy kapcsolatról van szó, akkor van jobb megoldás is, az amit ODL-nél csináltunk:
ha az E és F közti kapcsolat F felé egyirányú, akkor az E egyedhalmaz átírásakor bevesszük az F osztály kulcsát is. Ez ugyanazért lesz jó, miért az ODL-es és így eggyel kevesebb tábla lesz.
Így az előbbi E/K diagram esetén nem kell külön tábla a kapcsolatnak, hanem a

Film(cím, év, hossz, szalagFajta, **stúdióNév**)

lesz a Film tábla.

Gyenge egyedhalmazok kezelése

Ha *W* gyenge egyedhalmaz:

- Nem csak *W* attribútumait kell tartalmaznia, hanem azokat is, amikb?l *W* kulcsa el?áll. (Dupla keretes kapcsolat.)

Gyenge egyedhalmazok kezelése

Ha *W* gyenge egyedhalmaz:

- Nem csak *W* attribútumait kell tartalmaznia, hanem azokat is, amikb?l *W* kulcsa el?áll. (Dupla keretes kapcsolat.)
- Ez minden olyan kapcsolatra is igaz, melyben *W* részt vesz és amelyben így szerepel *W* kulcsa.

Gyenge egyedhalmazok kezelése

Ha *W* gyenge egyedhalmaz:

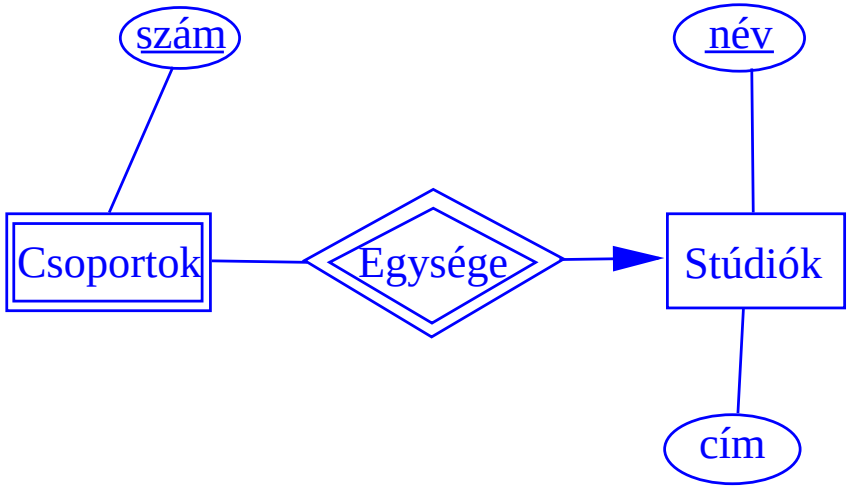
- Nem csak *W* attribútumait kell tartalmaznia, hanem azokat is, amikb?l *W* kulcsa el?áll. (Dupla keretes kapcsolat.)
- Ez minden olyan kapcsolatra is igaz, melyben *W* részt vesz és amelyben így szerepel *W* kulcsa.
- A dupla keretes kapcsolatokhoz nem kell külön reláció (mert az az infó már egyszer szerepel a gyenge egyedhalmaz megadásánál).

Gyenge egyedhalmazok kezelése

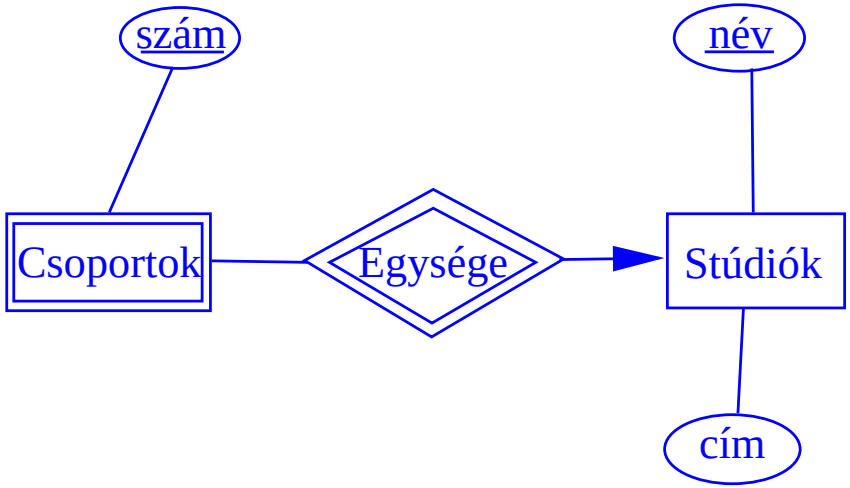
Ha *W* gyenge egyedhalmaz:

- Nem csak *W* attribútumait kell tartalmaznia, hanem azokat is, amikb?l *W* kulcsa el?áll. (Dupla keretes kapcsolat.)
- Ez minden olyan kapcsolatra is igaz, melyben *W* részt vesz és amelyben így szerepel *W* kulcsa.
- A dupla keretes kapcsolatokhoz nem kell külön reláció (mert az az infó már egyszer szerepel a gyenge egyedhalmaz megadásánál).

Gyenge egyedhalmazok kezelése

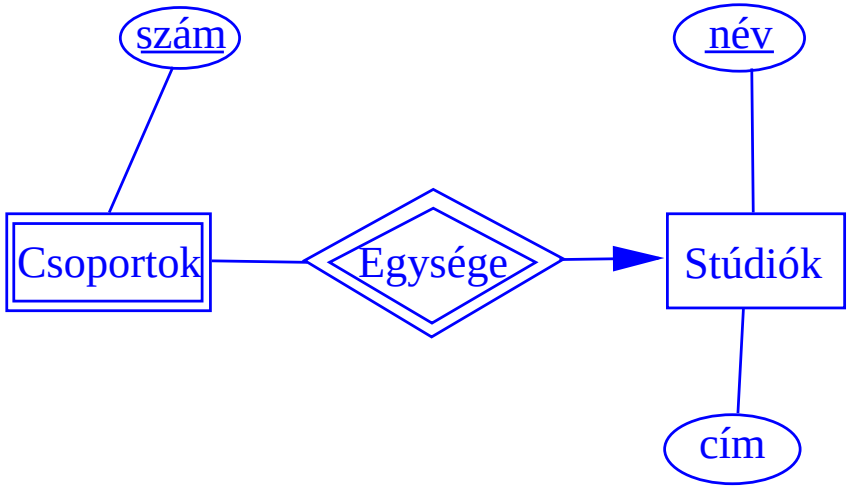


Gyenge egyedhalmazok kezelése



Stúdió(<u>név</u>, cím)
Csoportok(<u>szám</u>, <u>stúdióNév</u>)
Egység(szám, stúdióNév, név)

Gyenge egyedhalmazok kezelése

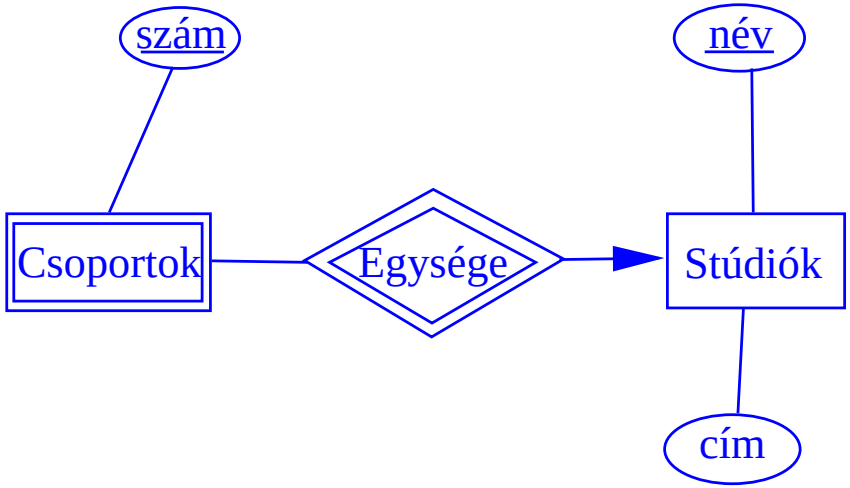


`Stúdió(név, cím)`
`Csoportok(szám, stúdióNév)`
`Egység(szám, stúdióNév, név)`

`Egység(szám, stúdióNév, név)` $\Longrightarrow$ `Egység(szám, név)` hiszen ugyanaz kétszer.

$\Longrightarrow$ `Egység` el is hagyható, hiszen összes attribútuma szerepel a `Csoport`-ban is.

Gyenge egyedhalmazok kezelése



`Stúdió(név, cím)`
`Csoportok(szám, stúdióNév)`
`Egység(szám, stúdióNév, név)`

`Egység(szám, stúdióNév, név)` $\Longrightarrow$ `Egység(szám, név)` hiszen ugyanaz kétszer.

$\Longrightarrow$ `Egység` el is hagyható, hiszen összes attribútuma szerepel a `Csoport`-ban is.

Ez általában is igaz, hiszen a gyenge egyedhalmaz attribútumai között ott lesz a dupla keretes rombusz minden attribútuma.

Alosztályok kezelése ODL-ben

`Film, Rajzfilm, Krimi, KrimiRajzfilm` esete. *(Itt minden egyed pont egy osztályban lehet benne, ezért kellett KrimiRajzfilm osztályt is megadni, pl. a Macskafogó miatt.)*

Alosztályok kezelése ODL-ben

`Film, Rajzfilm, Krimi, KrimiRajzfilm` esete. *(Itt minden egyed pont egy osztályban lehet benne, ezért kellett KrimiRajzfilm osztályt is megadni, pl. a Macskafogó miatt.)*
Minden alosztályhoz egy reláció, minden attribútumával és kapcsolatával (öröklöttekkel is).

Alosztályok kezelése ODL-ben

`Film, Rajzfilm, Krimi, KrimiRajzfilm` esete. *(Itt minden egyed pont egy osztályban lehet benne, ezért kellett KrimiRajzfilm osztályt is megadni, pl. a Macskafogó miatt.)*

Minden alosztályhoz egy reláció, minden attribútumával és kapcsolatával (öröklöttekkel is).

```
Film(cím, év, hossz, szalagFajta, stúdióNév, színésznév)
Rajzfilm(cím, év, hossz, szalagFajta, stúdióNév, színésznév,
hang)
BűnügyiFilm(cím, év, hossz, szalagFajta, stúdióNév, színésznév,
fegyver)
BűnügyiRajzfilm(cím, év, hossz, szalagFajta, stúdióNév,
színésznév, hang, fegyver)
```

Alosztályok kezelése ODL-ben

`Film, Rajzfilm, Krimi, KrimiRajzfilm` esete. *(Itt minden egyed pont egy osztályban lehet benne, ezért kellett KrimiRajzfilm osztályt is megadni, pl. a Macskafogó miatt.)*

Minden alosztályhoz egy reláció, minden attribútumával és kapcsolatával (öröklöttekkel is).

```
Film(cím, év, hossz, szalagFajta, stúdióNév, színésznév)
Rajzfilm(cím, év, hossz, szalagFajta, stúdióNév, színésznév,
hang)
BűnügyiFilm(cím, év, hossz, szalagFajta, stúdióNév, színésznév,
fegyver)
BűnügyiRajzfilm(cím, év, hossz, szalagFajta, stúdióNév,
színésznév, hang, fegyver)
```

Alosztályok kezelése ODL-ben

Kulcs: a főosztálynál úgy, ahogy eddig volt, az alosztály meg örökli a kulcsot, ha tudja

Alosztályok kezelése ODL-ben

Kulcs: a főosztálynál úgy, ahogy eddig volt, az alosztály meg örökli a kulcsot, ha tudja
Hátrány: egy film kereséséhez mind a négy relációt végig kell nézni, ha nem tudjuk, hogy hova tartozik a keresett film.

Alosztályok kezelése ODL-ben

Kulcs: a főosztálynál úgy, ahogy eddig volt, az alosztály meg örökli a kulcsot, ha tudja
Hátrány: egy film kereséséhez mind a négy relációt végig kell nézni, ha nem tudjuk, hogy hova tartozik a keresett film.
Megjegyzés: ebben a példában a kulcsok elromlanak a többes kapcsolatok miatt

Alosztályok kezelése E/K modellben

E/K-ban nem kell egy egyednek egyetlen egyedhalmazban lennie (ezért nem lesz itt KrimiRajzfilm egyedhalmaz), előfordulhat, hogy egy filmre vonatkozó információk szét vannak szórva.

Alosztályok kezelése E/K modellben

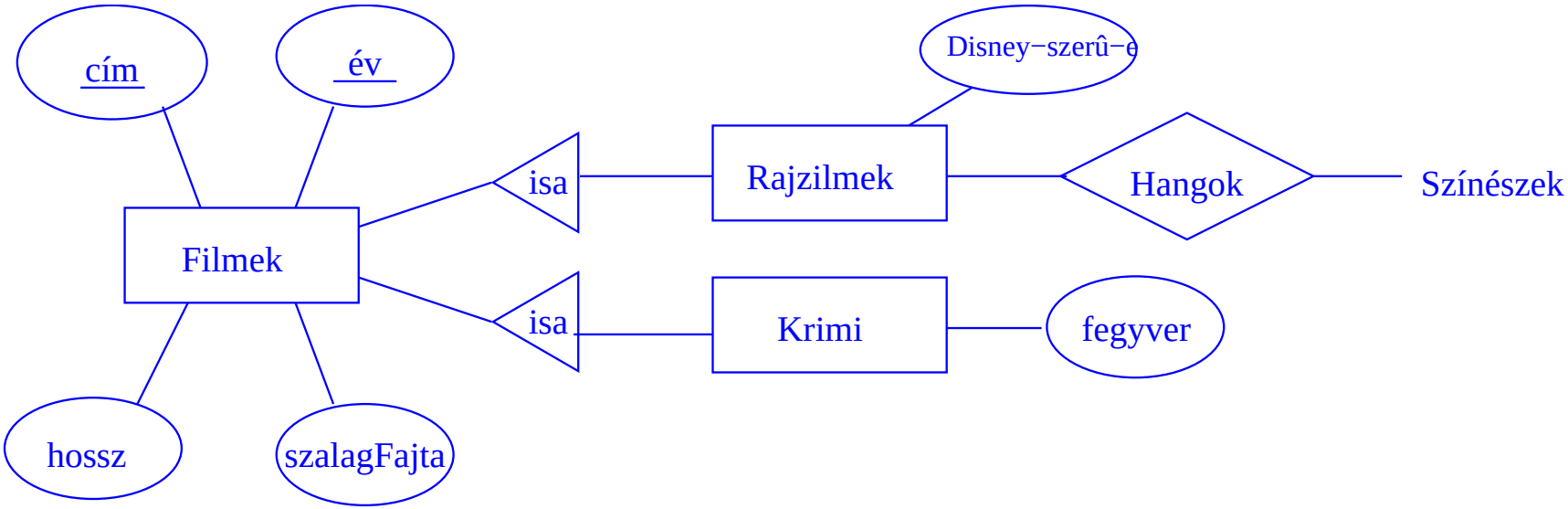
E/K-ban nem kell egy egyednek egyetlen egyedhalmazban lennie (ezért nem lesz itt KrimiRajzfilm egyedhalmaz), előfordulhat, hogy egy filmre vonatkozó információk szét vannak szórva.
A relációs sémára való átíráskor gondoskodunk róla, hogy a részinfókból vissza tudjuk állítani az egészet.
Átírás: Minden alosztályhoz csak a főosztály kulcsát és saját attribútumait rendeljük.

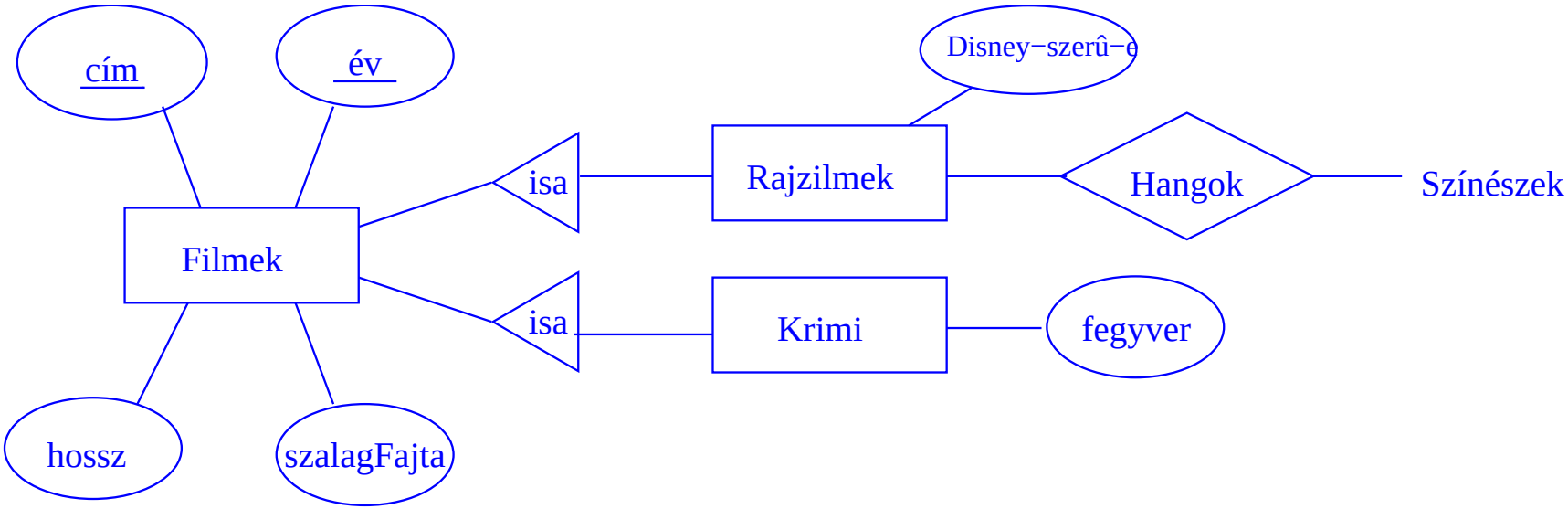
Alosztályok kezelése E/K modellben

E/K-ban nem kell egy egyednek egyetlen egyedhalmazban lennie (ezért nem lesz itt KrimiRajzfilm egyedhalmaz), előfordulhat, hogy egy filmre vonatkozó információk szét vannak szórva.
A relációs sémára való átíráskor gondoskodunk róla, hogy a részinfókból vissza tudjuk állítani az egészet.
Átírás: Minden alosztályhoz csak a főosztály kulcsát és saját attribútumait rendeljük. Az alosztály kulcsa a főosztály kulcsa lesz, így a kapcsolatba is ezt viszi magával az alosztály.

Alosztályok kezelése E/K modellben

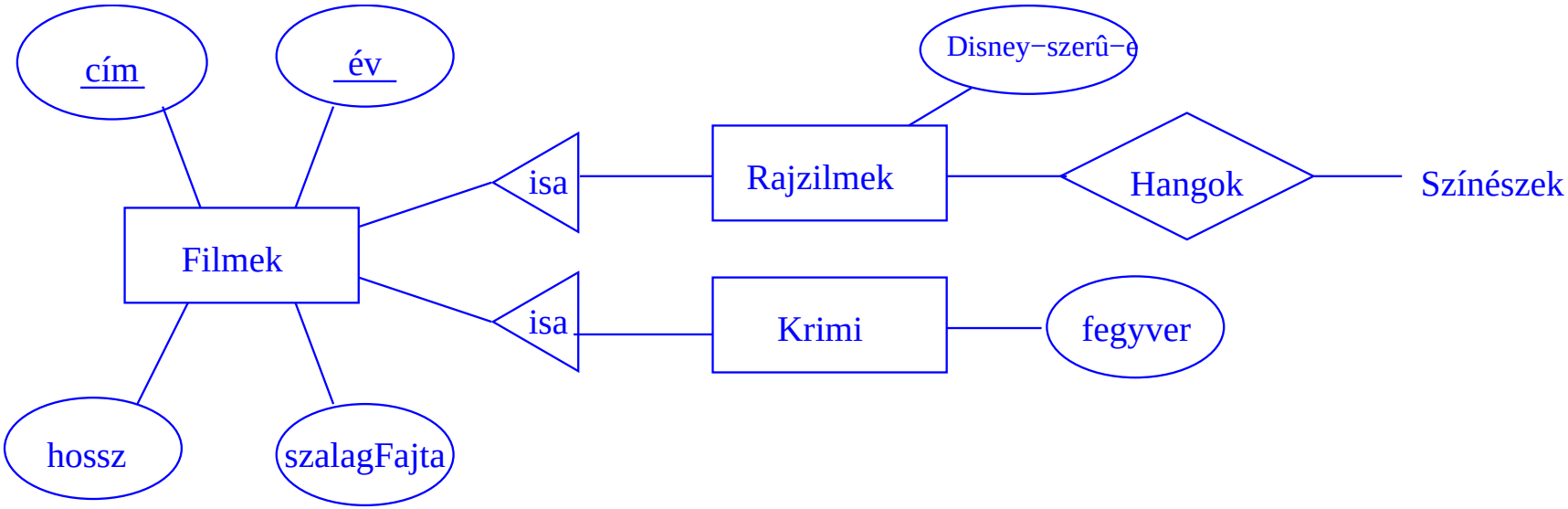
E/K-ban nem kell egy egyednek egyetlen egyedhalmazban lennie (ezért nem lesz itt KrimiRajzfilm egyedhalmaz), előfordulhat, hogy egy filmre vonatkozó információk szét vannak szórva.
A relációs sémára való átíráskor gondoskodunk róla, hogy a részinfókból vissza tudjuk állítani az egészet.
Átírás: Minden alosztályhoz csak a főosztály kulcsát és saját attribútumait rendeljük. Az alosztály kulcsa a főosztály kulcsa lesz, így a kapcsolatba is ezt viszi magával az alosztály.
Az „isa” kapcsolathoz nem rendelünk relációt.


cím
év
Disney−szerû−e
isa
Rajzilmek
Hangok
Színészek
Filmek
isa
Krimi
fegyver
hossz
szalagFajta



$\Longrightarrow$

Film(cím, év, hossz, szalagFajta)
Rajzfilm(cím, év, Disney-szerű-e)
Krimi(cím, év, fegyver)
Hangok(cím, év, Színésznév)



$\Longrightarrow$

`Film(cím, év, hossz, szalagFajta)`
`Rajzfilm(cím, év, Disney-szerű-e)`
`Krimi(cím, év, fegyver)`
`Hangok(cím, év, Színésznév)`

Hátránya: egy film információi több helyre vannak szórva (pl. Macskafogónál: a hossz és a szalagfajta a Film-ben, az, hogy nem Disney-is, az a Rajzfilmben, hangok a Hangokban. De ezeket az infókat össze lehet rakni, a (cím, év) kulcs menti természetes illesztéssel).

Másik megoldás NULL értékkel

`Film(cím, év, hossz, szalagFajta, Disney-szerű-e, színésznév, fegyver)`

Másik megoldás NULL értékkel

`Film(cím, év, hossz, szalagFajta, Disney-szerű-e, színésznév, fegyver)`

A hiányzó helyeket NULL-al töltjük ki.

Másik megoldás NULL értékkel

`Film(cím, év, hossz, szalagFajta, Disney-szerű-e, színésznév, fegyver)`

A hiányzó helyeket NULL-al töltjük ki.

Hátrány:

1. elveszíthetünk információt. Pl. egy olyan krimiről, amiben nincs fegyver, nem tudjuk, hogy krimi

Másik megoldás NULL értékkel

`Film(cím, év, hossz, szalagFajta, Disney-szerű-e, színésznév, fegyver)`

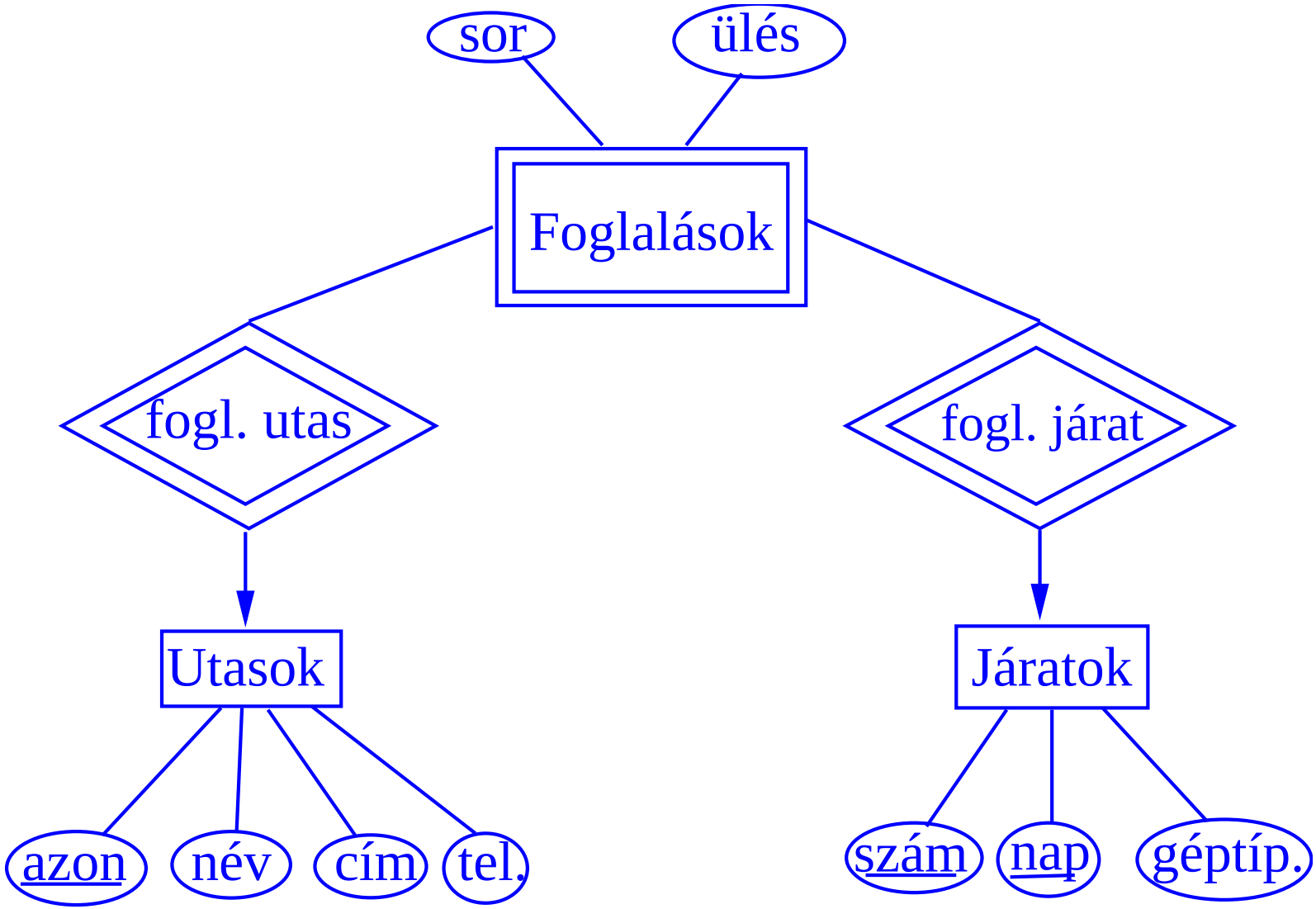
A hiányzó helyeket NULL-al töltjük ki.

Hátrány:

1. elveszíthetünk információt. Pl. egy olyan krimiről, amiben nincs fegyver, nem tudjuk, hogy krimi
2. a (cím, év) pár nem lesz kulcs, ugyanúgy, ahogy az ODL-es átírásnál sem lett

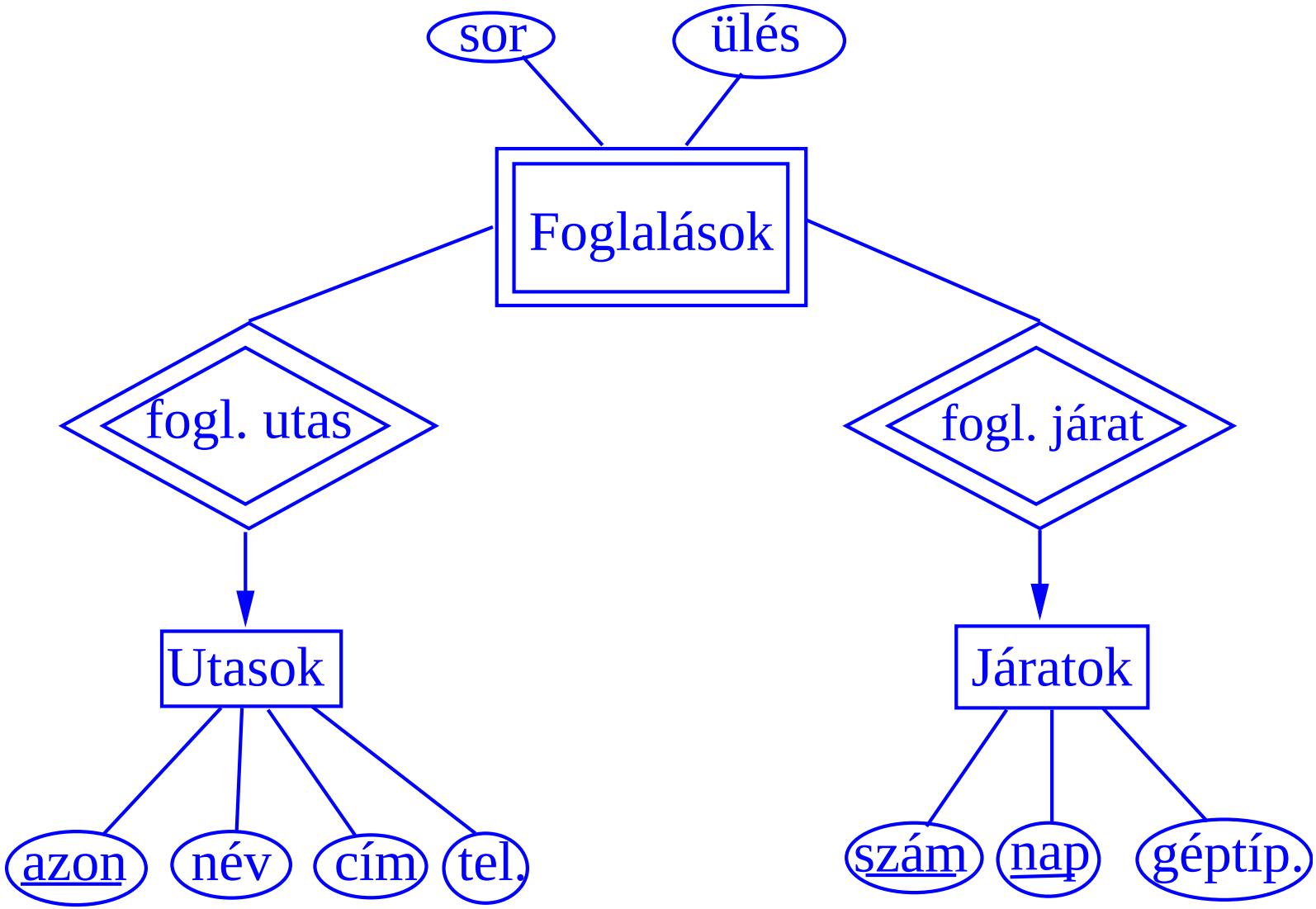
Példák

Alakítsuk relációssá:



Példák

Alakítsuk relációssá:



Megoldás

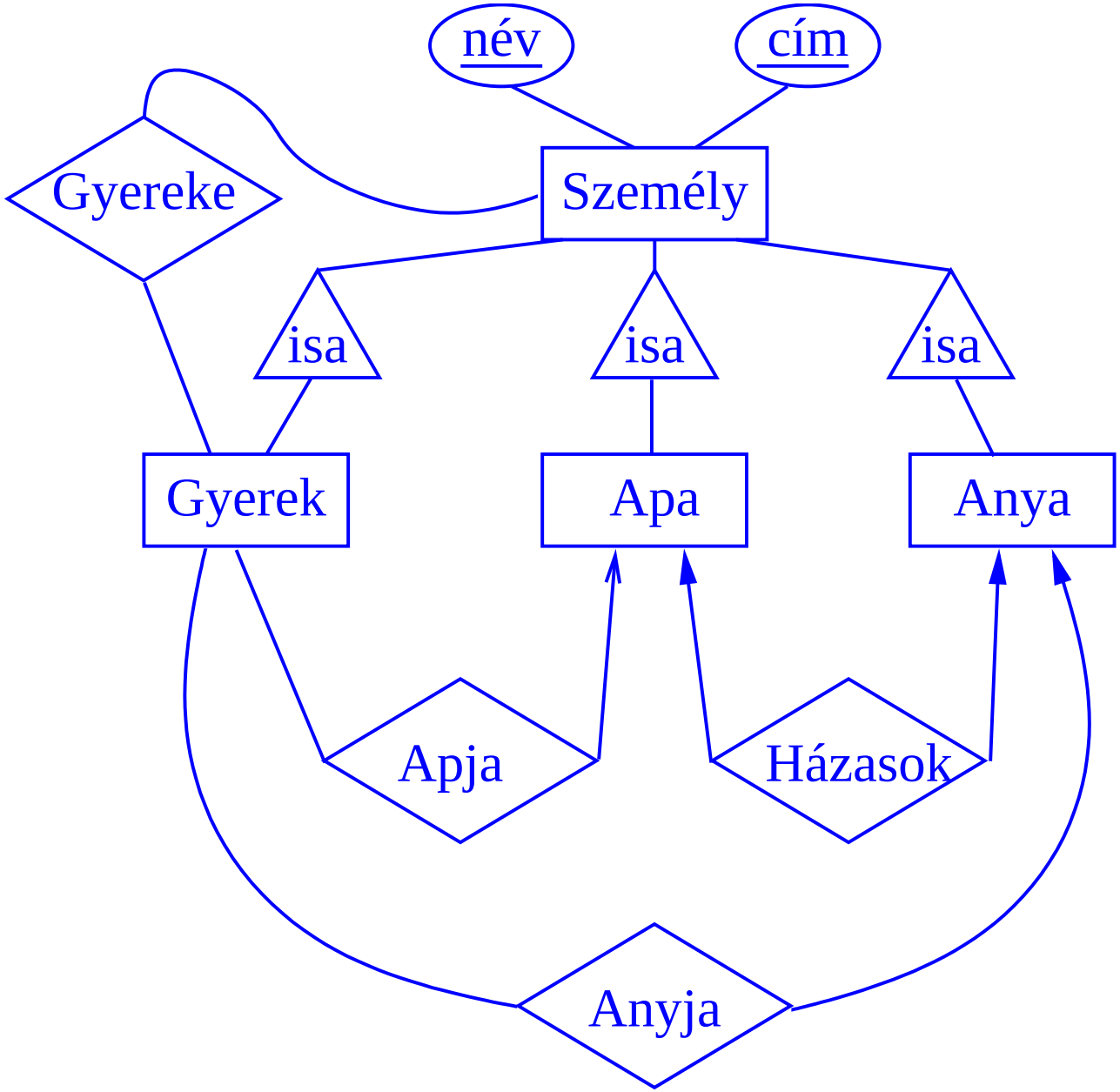
Utas(azon, név, cím, tel.)
Járat(szám, nap, géptípus)
Foglalások(azon,szám, nap, sor, ülés)

Megoldás

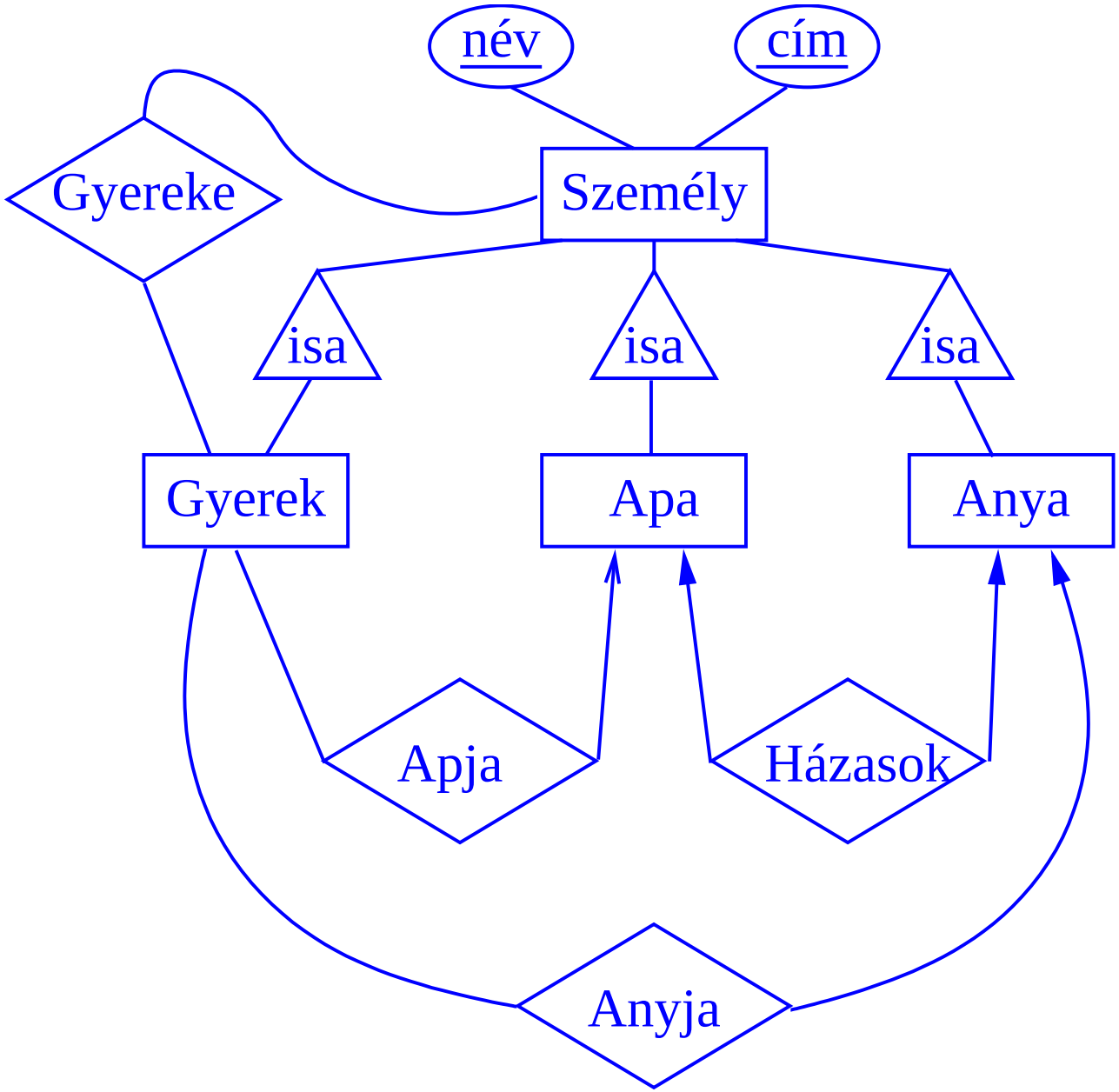
Utas(azon, név, cím, tel.)
Járat(szám, nap, géptípus)
Foglalások(azon,szám, nap, sor, ülés)

Megjegyzés: ha a Foglalások nem gyenge egyedhalmaz lett volna, hanem kapcsolat a két egyedhalmaz között, akkor is ugyanez a séma relációs jött volna ki.

Példák


név
cím
Gyereke
Személy
isa
isa
isa
Gyerek
Apa
Anya
Apja
Házasok
Anyja

Példák


név
cím
Gyereke
Személy
isa
isa
isa
Gyerek
Apa
Anya
Apja
Házasok
Anyja

Megoldás

Személy(<u>név</u>, <u>cím</u>); Gyerek(<u>név</u>, <u>cím</u>); Apa(<u>név</u>, <u>cím</u>)
Anya(<u>név</u>, <u>cím</u>)
Apja(<u>apanév</u>, <u>apacím</u>, <u>gyereknév</u>, <u>gyerekcím</u>)
Anyja(<u>anyanév</u>, <u>anyacím</u>, <u>gyereknév</u>, <u>gyerekcím</u>)
Házast(<u>fnév</u>, <u>fcím</u>, <u>nőnév</u>, <u>nőcím</u>)
Gyereke(<u>gyereknév</u>, <u>gyerekcím</u>, <u>szülőnév</u>, <u>szülőcím</u>)

Adatbázisok elmélete

Sorkalkulus

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

7. előadás

Sorkalkulus (Tuple calculus)

Lekérdezőnyelvek típusai:

- relációs algebra (**LEAP**, letölthető, SIGMOD-ról link), **ISBL**

Sorkalkulus (Tuple calculus)

Lekérdezőnyelvek típusai:

- relációs algebra (**LEAP**, letölthető, SIGMOD-ról link), **ISBL** *nehezen emészthetőbb, algebrai alapú; ez volt: láttuk, hogy relációs algebrával jól meg lehet adni relációkat*

Sorkalkulus (Tuple calculus)

Lekérdezőnyelvek típusai:

- relációs algebra (**LEAP**, letölthető, SIGMOD-ról link), **ISBL** *nehezen emészthetőbb, algebrai alapú; ez volt: láttuk, hogy relációs algebrával jól meg lehet adni relációkat*
- sorkalkulus (**SQL egy kicsit, QUEL**, INGRES nyelve),

Sorkalkulus (Tuple calculus)

Lekérdezőnyelvek típusai:

- relációs algebra (**LEAP**, letölthető, SIGMOD-ról link), **ISBL** *nehezen emészthetőbb, algebrai alapú; ez volt: láttuk, hogy relációs algebrával jól meg lehet adni relációkat*
- sorkalkulus (**SQL egy kicsit, QUEL**, INGRES nyelve), *könnyebben emészthető, logikai alapú*

Sorkalkulus (Tuple calculus)

Lekérdezőnyelvek típusai:

- relációs algebra (**LEAP**, letölthető, SIGMOD-ról link), **ISBL** *nehezen emészthetőbb, algebrai alapú; ez volt: láttuk, hogy relációs algebrával jól meg lehet adni relációkat*
- sorkalkulus (**SQL egy kicsit, QUEL**, INGRES nyelve), *könnyebben emészthető, logikai alapú*
- oszlopkalkulus (**QBE, SQL**) nagyon hasonló

Sorkalkulus (Tuple calculus)

Lekérdezőnyelvek típusai:

- relációs algebra (**LEAP**, letölthető, SIGMOD-ról link), **ISBL** *nehezen emészthetőbb, algebrai alapú; ez volt: láttuk, hogy relációs algebrával jól meg lehet adni relációkat*
- sorkalkulus (**SQL egy kicsit, QUEL**, INGRES nyelve), *könnyebben emészthető, logikai alapú*
- oszlopkalkulus (**QBE, SQL**) nagyon hasonló

Sorkalkulus

Formális modell, de már hasonlít az igazihoz.
Elsőrendű nyelv relációk kifejezésére.

Változók: t, r, s sorváltozók, a reláció sorainak felel meg

$t^{(k)}$: k oszlopos reláció sorainak felel meg

$t^{(k)}[i]$: A t sorváltozó i-edik komponense.

Pl. egy sor $\Longrightarrow$ (R. M., Budapest, hamburger, 180), akkor
$t^{(4)}[3] =$'hamburger' és $t^{(4)}[\text{ÁR}] = 180$

Sorkalkulus

Cél: sorkalkulussal relációkat megadni, úgy, mint relációs algebrával

Sorkalkulus

Cél: sorkalkulussal relációkat megadni, úgy, mint relációs algebrával
A sorkalkulus által kifejezett reláció:

$$\{t^{(k)} \mid \phi(t)\}$$

$\Longrightarrow$ a kifejezett reláció azon t-kből áll, amikre $\phi(t)$ igaz, ahol ϕ egy megengedett formula + valami még.

Sorkalkulus

Cél: sorkalkulussal relációkat megadni, úgy, mint relációs algebrával
A sorkalkulus által kifejezett reláció:

$$\{t^{(k)} \mid \phi(t)\}$$

$\Longrightarrow$ a kifejezett reláció azon t-kből áll, amikre $\phi(t)$ igaz, ahol ϕ egy megengedett formula + valami még.

Megengedett formulák (amik a $\phi(t)$ helyén állhatnak):

atomok:

- $R^{(k)}(t^{(k)})$: *(ahol R alapreláció), akkor igaz, ha t $\in$ R, azaz a sor benne van a relációban.*

Sorkalkulus

Cél: sorkalkulussal relációkat megadni, úgy, mint relációs algebrával
A sorkalkulus által kifejezett reláció:

$$\{t^{(k)} \mid \phi(t)\}$$

$\Longrightarrow$ a kifejezett reláció azon t-kből áll, amikre $\phi(t)$ igaz, ahol ϕ egy megengedett formula + valami még.

Megengedett formulák (amik a $\phi(t)$ helyén állhatnak):

atomok:

- $R^{(k)}(t^{(k)})$: *(ahol R alapreláció), akkor igaz, ha t $\in$ R, azaz a sor benne van a relációban.*
- ▶ $t^{(k)}[i]\ \theta\ s^{(l)}[j]$

Sorkalkulus

Cél: sorkalkulussal relációkat megadni, úgy, mint relációs algebrával
A sorkalkulus által kifejezett reláció:

$$\{t^{(k)} \mid \phi(t)\}$$

$\Longrightarrow$ a kifejezett reláció azon t-kből áll, amikre $\phi(t)$ igaz, ahol ϕ egy megengedett formula + valami még.

Megengedett formulák (amik a $\phi(t)$ helyén állhatnak):

atomok:

- $R^{(k)}(t^{(k)})$: *(ahol R alapreláció), akkor igaz, ha t $\in$ R, azaz a sor benne van a relációban.*
-
  - $t^{(k)}[i]\ \theta\ s^{(l)}[j]$
  - $t^{(k)}[i]\ \theta\ c$

Sorkalkulus

Cél: sorkalkulussal relációkat megadni, úgy, mint relációs algebrával
A sorkalkulus által kifejezett reláció:

$$\{t^{(k)} \mid \phi(t)\}$$

$\Longrightarrow$ a kifejezett reláció azon t-kből áll, amikre $\phi(t)$ igaz, ahol ϕ egy megengedett formula + valami még.

Megengedett formulák (amik a $\phi(t)$ helyén állhatnak):

atomok:

- $R^{(k)}(t^{(k)})$: *(ahol R alapreláció), akkor igaz, ha t $\in$ R, azaz a sor benne van a relációban.*
-
  - $t^{(k)}[i]\ \theta\ s^{(l)}[j]$
  - $t^{(k)}[i]\ \theta\ c$
  - $c\ \theta\ t^{(k)}[i]$

Sorkalkulus

Cél: sorkalkulussal relációkat megadni, úgy, mint relációs algebrával
A sorkalkulus által kifejezett reláció:

$$\{t^{(k)} \mid \phi(t)\}$$

$\Longrightarrow$ a kifejezett reláció azon t-kből áll, amikre $\phi(t)$ igaz, ahol ϕ egy megengedett formula + valami még.

Megengedett formulák (amik a $\phi(t)$ helyén állhatnak):

atomok:

- $R^{(k)}(t^{(k)})$: *(ahol R alapreláció), akkor igaz, ha t $\in$ R, azaz a sor benne van a relációban.*
- - $t^{(k)}[i]\ \theta\ s^{(l)}[j]$
  - $t^{(k)}[i]\ \theta\ c$
  - $c\ \theta\ t^{(k)}[i]$

  ahol $\theta \in \{<, >, =, \neq, \leq, \geq\}$, t, s sorváltozók, c konstans érték. *Világos, mikor igaz.*

építkezési szabályok:

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák.
  Világos, hogy mikor igaz.

építkezési szabályok:

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák.
  Világos, hogy mikor igaz.
- ϕ formula, s sorváltozó, akkor $\forall s\phi, \exists s\phi$ is formula.
  Világos, hogy mikor igaz.

építkezési szabályok:

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák.
  Világos, hogy mikor igaz.
- ϕ formula, s sorváltozó, akkor $\forall s\phi, \exists s\phi$ is formula.
  Világos, hogy mikor igaz.
  Kötött változó: ha vonatkozik rá kvantor,

építkezési szabályok:

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák.
  Világos, hogy mikor igaz.
- ϕ formula, s sorváltozó, akkor $\forall s\phi, \exists s\phi$ is formula.
  Világos, hogy mikor igaz.
  Kötött változó: ha vonatkozik rá kvantor,
  Szabad változó: ha nem,

építkezési szabályok:

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák.
  Világos, hogy mikor igaz.
- ϕ formula, s sorváltozó, akkor $\forall s\phi, \exists s\phi$ is formula.
  Világos, hogy mikor igaz.
  Kötött változó: ha vonatkozik rá kvantor,
  Szabad változó: ha nem,

Sorkalkulus által kifejezett reláció (pontosan):

$$\{t^{(k)} \mid \phi(t)\}$$

$\Longrightarrow$ a kifejezett reláció azon k hosszú t vektorokból áll, amikre $\phi(t)$ igaz, ahol ϕ egy megengedett formula és ϕ-*ben* t *az egyetlen szabad* változó.

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

$$\left\{ t^{(2)} \mid \text{BEVÉTEL}(t) \wedge t[1] \geq \text{2004-01-01} \right\}$$

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

$$\left\{ t^{(2)} \mid \text{BEVÉTEL}(t) \wedge t[1] \geq \text{2004-01-01} \right\}$$

Az 2004. jan. 15-i bevétel és a befizetett összeg:

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\right) \bowtie \text{BEFIZ}\right)$$

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

$$\left\{t^{(2)} \mid \text{BEVÉTEL}(t) \wedge t[1] \geq 2004\text{-}01\text{-}01\right\}$$

Az 2004. jan. 15-i bevétel és a befizetett összeg:

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\right) \bowtie \text{BEFIZ}\right)$$

$$\left\{u^{(2)} \mid \text{BEFIZ}(u) \wedge \exists v(\text{BEVÉTEL}(v) \wedge v[1] = 2004\text{-}01\text{-}15 \wedge v[2] = u[1])\right\}$$

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt:

$$\sigma_{\text{DÁTUM>'2004-01-01'}}\left(\text{BEVÉTEL}\right)$$

$$\left\{t^{(2)} \mid \text{BEVÉTEL}(t) \wedge t[1] \geq 2004\text{-}01\text{-}01\right\}$$

Az 2004. jan. 15-i bevétel és a befizetett összeg:

$$\pi_{\text{ÖSSZEG, BEFIZ}}\left(\sigma_{\text{DÁTUM='2004-01-15'}}\left(\text{BEVÉTEL}\right) \bowtie \text{BEFIZ}\right)$$

$$\left\{u^{(2)} \mid \text{BEFIZ}(u) \wedge \exists v(\text{BEVÉTEL}(v) \wedge v[1] = 2004\text{-}01\text{-}15 \wedge v[2] = u[1])\right\}$$

Példák sorkalkulus alkalmazására

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}} \left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2004-01-15'}} \left(\text{MENNYISÉG} \bowtie \text{ÁRU} \right) \right)$$

Példák sorkalkulus alkalmazására

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{ DÁTUM='2004-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right)$$

$$\left\{ s^{(3)} \mid \exists u \exists v \Big(\text{MENNYISÉG}(u) \wedge \text{ÁRU}(v) \wedge u[1] = \text{2004-01-15} \wedge u[2] =' A123' \wedge v[1] =' A123' \wedge s[1] = u[3] \wedge s[2] = v[2] \wedge s[3] = v[3] \Big) \right\}$$

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Mely nevű áruk azok, amelyekkel van azonos egységárú másik áru?

Példák sorkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Mely nevű áruk azok, amelyekkel van azonos egységárú másik áru?

$$\left\{ s^{(1)} \mid \exists u \exists v \left(\text{ÁRU}(v) \wedge \text{ÁRU}(u) \wedge s[1] = v[2] \wedge v[3] = u[3] \wedge \neg(v[1] = u[1]) \right) \right\}$$

Sorkalkulus ereje

Mivel lehet több mindent kifejezni?
Relációs algebrával, vagy sorkalkulussal?

Tétel

A sorkalkulus relációsan teljes.

Bizonyítás.

Be kell látni, hogy minden reláció, ami relációs algebrával megadható, megadható sorkalkulussal is. Ehhez azt elég megmutatni, hogy

Sorkalkulus ereje

Mivel lehet több mindent kifejezni?
Relációs algebrával, vagy sorkalkulussal?

Tétel

A sorkalkulus relációsan teljes.

Bizonyítás.

Be kell látni, hogy minden reláció, ami relációs algebrával megadható, megadható sorkalkulussal is. Ehhez azt elég megmutatni, hogy
1. az alaprelációk megadhatók

Sorkalkulus ereje

Mivel lehet több mindent kifejezni?
Relációs algebrával, vagy sorkalkulussal?

Tétel

A sorkalkulus relációsan teljes.

Bizonyítás.

Be kell látni, hogy minden reláció, ami relációs algebrával megadható, megadható sorkalkulussal is. Ehhez azt elég megmutatni, hogy
1. az alaprelációk megadhatók
2. a relációs algebrai alapműveletek (unió, különbség, szorzat, vetítés, szelekció) alaprelációkra alkalmazva megvalósíthatók

Sorkalkulus ereje

Mivel lehet több mindent kifejezni?
Relációs algebrával, vagy sorkalkulussal?

Tétel

A sorkalkulus relációsan teljes.

Bizonyítás.

Be kell látni, hogy minden reláció, ami relációs algebrával megadható, megadható sorkalkulussal is. Ehhez azt elég megmutatni, hogy
1. az alaprelációk megadhatók
2. a relációs algebrai alapműveletek (unió, különbség, szorzat, vetítés, szelekció) alaprelációkra alkalmazva megvalósíthatók
3. ha R és S nem alapreláció és ezekre alkalmazunk valami relációs alapműveletet, akkor az eredmény kifejezhető sorkalkulussal

Bizonyítás.

- ***alapreláció:*** *Tegyük fel, hogy R k oszlopos alapreláció* $R = \left\{ t^{(k)} \mid R(t) \right\}$

Bizonyítás.

- ***alapreláció:*** *Tegyük fel, hogy R k oszlopos alapreláció*
  $R = \left\{ t^{(k)} \mid R(t) \right\}$
- ***unió:*** *Tfh. S is k oszlopos*
  $R \cup S = \left\{ t^{(k)} \mid R(t) \vee S(t) \right\}$

Bizonyítás.

- ***alapreláció:*** *Tegyük fel, hogy R k oszlopos alapreláció*
  $R = \left\{ t^{(k)} \mid R(t) \right\}$
- ***unió:*** *Tfh. S is k oszlopos*
  $R \cup S = \left\{ t^{(k)} \mid R(t) \vee S(t) \right\}$
- ***különbség:***
  $R \setminus S = \left\{ t^{(k)} \mid R(t) \wedge \neg S(t) \right\}$

Bizonyítás.

- ***alapreláció:*** *Tegyük fel, hogy R k oszlopos alapreláció*
  $R = \left\{ t^{(k)} \mid R(t) \right\}$
- ***unió:*** *Tfh. S is k oszlopos*
  $R \cup S = \left\{ t^{(k)} \mid R(t) \vee S(t) \right\}$
- ***különbség:***
  $R \setminus S = \left\{ t^{(k)} \mid R(t) \wedge \neg S(t) \right\}$
- *metszet:*
  $R \cap S = \left\{ t^{(k)} \mid R(t) \wedge S(t) \right\}$

Bizonyítás.

- ***alapreláció:*** *Tegyük fel, hogy R k oszlopos alapreláció*
  $R = \left\{ t^{(k)} \mid R(t) \right\}$
- ***unió:*** *Tfh. S is k oszlopos*
  $R \cup S = \left\{ t^{(k)} \mid R(t) \vee S(t) \right\}$
- ***különbség:***
  $R \setminus S = \left\{ t^{(k)} \mid R(t) \wedge \neg S(t) \right\}$
- *metszet:*
  $R \cap S = \left\{ t^{(k)} \mid R(t) \wedge S(t) \right\}$
- ***szorzat:*** *R legyen k oszlopos, S pedig l oszlopos*

$$R \times S = \left\{ t^{(k+l)} \mid \exists r^{(k)} \exists s^{(l)} \left(R(r) \wedge S(s) \wedge r[1] = t[1] \wedge \ldots \right. \right.$$
$$\left. \left. \wedge r[k] = t[k] \wedge s[1] = t[k+1] \wedge \ldots \wedge s[l] = t[k+l] \right) \right\}$$

Bizonyítás.

- ***alapreláció:*** *Tegyük fel, hogy R k oszlopos alapreláció*
  $R = \left\{ t^{(k)} \mid R(t) \right\}$
- ***unió:*** *Tfh. S is k oszlopos*
  $R \cup S = \left\{ t^{(k)} \mid R(t) \vee S(t) \right\}$
- ***különbség:***
  $R \setminus S = \left\{ t^{(k)} \mid R(t) \wedge \neg S(t) \right\}$
- *metszet:*
  $R \cap S = \left\{ t^{(k)} \mid R(t) \wedge S(t) \right\}$
- ***szorzat:*** *R legyen k oszlopos, S pedig l oszlopos*

$$R \times S = \left\{ t^{(k+l)} \mid \exists r^{(k)} \exists s^{(l)} \left(R(r) \wedge S(s) \wedge r[1] = t[1] \wedge \ldots \right. \right.$$
$$\left. \left. \wedge r[k] = t[k] \wedge s[1] = t[k+1] \wedge \ldots \wedge s[l] = t[k+l] \right) \right\}$$

- ***vetület:*** *Legyen $R(A_1, \ldots, A_d, A_{d+1}, \ldots, A_k)$ reláció, vetítsük az első d-re*
  $\pi_{A_1,\ldots,A_d}(R) = \left\{ t^{(d)} \mid \exists r^{(k)} \left(R(r) \wedge r[1] = t[1] \wedge \ldots \wedge r[d] = t[d] \right) \right\}$

Bizonyítás.

- **kiválasztás:**
  $\sigma_F(R) = \left\{ t^{(k)} \mid R(t) \wedge F' \right\}$, ahol F' átfordítása sorkalkulusra $\Longrightarrow$ az i-edik attribútum helyett $t^{(n)}[i]$-t írunk.
  Pl. (evidenciával történő meggyőzés)
  $\sigma_{\text{ÁR>'150'} \wedge \text{TERMÉK='hamburger'}}(\text{TERMEL}) =$
  $\left\{ t^{(4)} \mid \text{TERMEL}(t) \wedge t[4] > \text{'150'} \wedge t[3] = \text{'hamburger'} \right\}$

Bizonyítás.

- **kiválasztás:**
  $\sigma_F(R) = \left\{ t^{(k)} \mid R(t) \wedge F' \right\}$, ahol F' átfordítása sorkalkulusra $\Longrightarrow$ az i-edik attribútum helyett $t^{(n)}[i]$-t írunk.
  Pl. (evidenciával történő meggyőzés)
  $\sigma_{\text{ÁR>'150'} \wedge \text{TERMÉK='hamburger'}}(\text{TERMEL}) =$
  $\left\{ t^{(4)} \mid \text{TERMEL}(t) \wedge t[4] > \text{'150'} \wedge t[3] = \text{'hamburger'} \right\}$
- Nem lényeges, hogy R, S alaprelációk.
  Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ és $S = \left\{ t^{(k)} \mid \psi(t) \right\}$, azaz R és S már valahogy ki van fejezve sorkalkulussal
  $\Longrightarrow R \cup S = \left\{ t^{(k)} \mid \phi(t) \vee \psi(t) \right\}$
  többinél ugyanígy ($R(t)$ és $S(t)$ helyett $\phi(t)$-t és $\psi(t)$-t írunk). $\surd$

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Bizonyítás.

Relációs algebrában minden reláció véges, *ha az alaprelációk végesek.* Ez viszont lehet végtelen, ha az egyik értékkészlet végtelen. $\surd$

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Bizonyítás.

Relációs algebrában minden reláció véges, *ha az alaprelációk végesek.* Ez viszont lehet végtelen, ha az egyik értékkészlet végtelen. $\surd$ □

Alkalmazásokban, bár elvileg minden véges, gyakorlatban az is problémát okoz, ha nagyon nagy méret?. Ez ténylegesen el? is fodrulhat, erre vigyázni kell.

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Bizonyítás.

Relációs algebrában minden reláció véges, *ha az alaprelációk végesek.* Ez viszont lehet végtelen, ha az egyik értékkészlet végtelen. $\surd$ □

Alkalmazásokban, bár elvileg minden véges, gyakorlatban az is problémát okoz, ha nagyon nagy méret?. Ez ténylegesen el? is fodrulhat, erre vigyázni kell.
Sőt, még ha az eredmény mérete nem is túl nagy, részeredményekben lehet nagyon nagy tábla $\Longrightarrow$ túl sok munka.

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Bizonyítás.

Relációs algebrában minden reláció véges, *ha az alaprelációk végesek.* Ez viszont lehet végtelen, ha az egyik értékkészlet végtelen. $\surd$ □

Alkalmazásokban, bár elvileg minden véges, gyakorlatban az is problémát okoz, ha nagyon nagy méret?. Ez ténylegesen el? is fodrulhat, erre vigyázni kell.
Sőt, még ha az eredmény mérete nem is túl nagy, részeredményekben lehet nagyon nagy tábla $\Longrightarrow$ túl sok munka.

Megoldás:

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Bizonyítás.

Relációs algebrában minden reláció véges, *ha az alaprelációk végesek.* Ez viszont lehet végtelen, ha az egyik értékkészlet végtelen. $\surd$ □

Alkalmazásokban, bár elvileg minden véges, gyakorlatban az is problémát okoz, ha nagyon nagy méret?. Ez ténylegesen el? is fodrulhat, erre vigyázni kell.
Sőt, még ha az eredmény mérete nem is túl nagy, részeredményekben lehet nagyon nagy tábla $\Longrightarrow$ túl sok munka.

Megoldás: Nem használunk ilyesmit $\Longrightarrow$ csak **biztonságos formulákat** (safe expression):

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Bizonyítás.

Relációs algebrában minden reláció véges, *ha az alaprelációk végesek.* Ez viszont lehet végtelen, ha az egyik értékkészlet végtelen. $\surd$ □

Alkalmazásokban, bár elvileg minden véges, gyakorlatban az is problémát okoz, ha nagyon nagy méret?. Ez ténylegesen el? is fodrulhat, erre vigyázni kell.
Sőt, még ha az eredmény mérete nem is túl nagy, részeredményekben lehet nagyon nagy tábla $\Longrightarrow$ túl sok munka.

Megoldás: Nem használunk ilyesmit $\Longrightarrow$ csak **biztonságos formulákat** (safe expression): kiértékelhető úgy, hogy ne kelljen túl nagy halmazt végignézni, csak annyi infó kell hozzá, amit valaki már egyszer korábban beírt

Megfordítás

Ki lehet-e fejezni mindet relációs algebrával, amit sorkalkulussal lehet?

Nem!

Pl. Ha R egy k változós alapreláció $\Longrightarrow \left\{ t^{(k)} \mid \neg R(t) \right\}$ nem fejezhető ki algebrával.

Bizonyítás.

Relációs algebrában minden reláció véges, *ha az alaprelációk végesek.* Ez viszont lehet végtelen, ha az egyik értékkészlet végtelen. $\surd$ □

Alkalmazásokban, bár elvileg minden véges, gyakorlatban az is problémát okoz, ha nagyon nagy méret?. Ez ténylegesen el? is fodrulhat, erre vigyázni kell.
Sőt, még ha az eredmény mérete nem is túl nagy, részeredményekben lehet nagyon nagy tábla $\Longrightarrow$ túl sok munka.

Megoldás: Nem használunk ilyesmit $\Longrightarrow$ csak **biztonságos formulákat** (safe expression): kiértékelhető úgy, hogy ne kelljen túl nagy halmazt végignézni, csak annyi infó kell hozzá, amit valaki már egyszer korábban beírt $\Longrightarrow$
leszűkítjük a szóba jövő esetek halmazát

Definíció

Dom(ϕ)= {ϕ-*beli alaprelációk* $\forall$ *attribútumának,* $\forall$ *értéke*} $\cup$ {ϕ-*beli konstansok*}

Definíció

Dom(ϕ)= {ϕ-*beli alaprelációk* $\forall$ *attribútumának,* $\forall$ *értéke*} $\cup$ {ϕ-*beli konstansok*}

Pl. SZEMÉLY(NÉV, CÍM) alapreláció

$\phi(t) = \text{SZEMÉLY}(t) \wedge t[2] = \text{'Tokyo'}$

$\Longrightarrow Dom(\phi) = \pi_{\text{NÉV}}\left(\text{SZEMÉLY}\right) \cup \pi_{\text{CÍM}}\left(\text{SZEMÉLY}\right) \cup \{\text{'Tokyo'}\}$

Definíció

Dom(ϕ)= {ϕ-*beli alaprelációk* $\forall$ *attribútumának,* $\forall$ *értéke*} $\cup$ {ϕ-*beli konstansok*}

Pl. SZEMÉLY(NÉV, CÍM) alapreláció

$\phi(t) = \text{SZEMÉLY}(t) \wedge t[2] = \text{'Tokyo'}$

$\Longrightarrow Dom(\phi) = \pi_{\text{NÉV}}\left(\text{SZEMÉLY}\right) \cup \pi_{\text{CÍM}}\left(\text{SZEMÉLY}\right) \cup \{\text{'Tokyo'}\}$

Biztonságos reláció

Definíció

Egy $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ ***reláció biztonságos****, ha*

i) *Minden* $\phi(t)$*-t kielégítő* t *minden komponense* $\in$ *Dom*(ϕ)
(Ha t *kielégíti* $\phi(t)$*-t, akkor minden komponense Dom*(ϕ)*-beli)*
(Ez korlátozza keresést! A végeredménybe csak Dom(ϕ)*-ből lehet kerülni.)*

Biztonságos reláció

Definíció

Egy $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ ***reláció biztonságos****, ha*

i) *Minden* $\phi(t)$*-t kielégítő* t *minden komponense* $\in$ *Dom*(ϕ)
(Ha t *kielégíti* $\phi(t)$*-t, akkor minden komponense Dom*(ϕ)*-beli)*
(Ez korlátozza keresést! A végeredménybe csak Dom(ϕ)*-ből lehet kerülni.)*

ii) ϕ *minden* $\exists u \psi(u)$ *alakú részformulájára igaz, hogy ha* u *kielégíti* ψ*-t a* ψ*-beli szabad változók valamely értékeire, akkor* u *minden komponense* $\in$ *Dom*(ψ)
(részformula is biztonságos: $\exists u \psi(u)$ *igazsága eldönthető Dom*(ψ) *végignézésével)*

Biztonságos reláció

Definíció

Egy $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ ***reláció biztonságos****, ha*

i) *Minden* $\phi(t)$*-t kielégítő* t *minden komponense* $\in Dom(\phi)$
(Ha t *kielégíti* $\phi(t)$*-t, akkor minden komponense* $Dom(\phi)$*-beli)*
(Ez korlátozza keresést! A végeredménybe csak $Dom(\phi)$*-ből lehet kerülni.)*

ii) ϕ *minden* $\exists u\psi(u)$ *alakú részformulájára igaz, hogy ha* u *kielégíti* ψ*-t a* ψ*-beli szabad változók valamely értékeire, akkor* u *minden komponense* $\in Dom(\psi)$
(részformula is biztonságos: $\exists u\psi(u)$ *igazsága eldönthető* $Dom(\psi)$ *végignézésével)*

Megj.: *A* $\forall u\psi(u)$ *alakúakra nem kell, mert ez ugyanaz, mint* $\neg\exists u(\neg\psi(u))$ *és így elég (ii)-t ellenőrizni a* $\exists u(\neg\psi(u))$ *részformulára.*

Biztonságos reláció

Definíció

Egy $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ ***reláció biztonságos****, ha*

i) *Minden* $\phi(t)$*-t kielégítő* t *minden komponense* $\in Dom(\phi)$
(Ha t *kielégíti* $\phi(t)$*-t, akkor minden komponense* $Dom(\phi)$*-beli)*
(Ez korlátozza keresést! A végeredménybe csak $Dom(\phi)$*-ből lehet kerülni.)*

ii) ϕ *minden* $\exists u\psi(u)$ *alakú részformulájára igaz, hogy ha* u *kielégíti* ψ*-t a* ψ*-beli szabad változók valamely értékeire, akkor* u *minden komponense* $\in Dom(\psi)$
(részformula is biztonságos: $\exists u\psi(u)$ *igazsága eldönthető* $Dom(\psi)$ *végignézésével)*

Megj.: *A* $\forall u\psi(u)$ *alakúakra nem kell, mert ez ugyanaz, mint* $\neg\exists u(\neg\psi(u))$ *és így elég (ii)-t ellenőrizni a* $\exists u(\neg\psi(u))$ *részformulára.*

Nem az a kérdés, hogy pontosan mik a biztonságos formulák, hanem:

Biztonságos reláció

Definíció

Egy $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ ***reláció biztonságos****, ha*

i) *Minden* $\phi(t)$*-t kielégítő* t *minden komponense* $\in Dom(\phi)$
(Ha t *kielégíti* $\phi(t)$*-t, akkor minden komponense* $Dom(\phi)$*-beli)*
(Ez korlátozza keresést! A végeredménybe csak $Dom(\phi)$*-ből lehet kerülni.)*

ii) ϕ *minden* $\exists u \psi(u)$ *alakú részformulájára igaz, hogy ha* u *kielégíti* ψ*-t a* ψ*-beli szabad változók valamely értékeire, akkor* u *minden komponense* $\in Dom(\psi)$
(részformula is biztonságos: $\exists u \psi(u)$ *igazsága eldönthető* $Dom(\psi)$ *végignézésével)*

Megj.: *A* $\forall u \psi(u)$ *alakúakra nem kell, mert ez ugyanaz, mint* $\neg \exists u(\neg \psi(u))$ *és így elég (ii)-t ellenőrizni a* $\exists u(\neg \psi(u))$ *részformulára.*

Nem az a kérdés, hogy pontosan mik a biztonságos formulák, hanem:
Hogyan tudunk biztonságos formulákat, kifejezéseket írni?
Csak ilyeneket szeretnénk használni.

Biztonságos formulák

Tipikus technikák a biztonságosság elérésére:

- $\{t \mid R(t) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos (mert szűkítés R-re, így (i) teljesül)

Biztonságos formulák

Tipikus technikák a biztonságosság elérésére:

- $\{t \mid R(t) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos (mert szűkítés R-re, így (i) teljesül)
- *(Ha nem lehet egy relációra korlátozni pl:)*
  $\{t \mid (R(t) \vee S(t)) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos

Biztonságos formulák

Tipikus technikák a biztonságosság elérésére:

- $\{t \mid R(t) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos (mert szűkítés R-re, így (i) teljesül)
- *(Ha nem lehet egy relációra korlátozni pl:)*
  $\{t \mid (R(t) \vee S(t)) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos
- $\left\{t^{(2)} \mid \exists u_1 \exists u_2 \left(R(u_1) \wedge R(u_2) \wedge t[1] = u_2[1] \wedge t[2] = u_1[1] \wedge \phi(t, u_1, u_2)\right)\right\}$

Biztonságos formulák

Tipikus technikák a biztonságosság elérésére:

- $\{t \mid R(t) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos (mert szűkítés R-re, így (i) teljesül)
- *(Ha nem lehet egy relációra korlátozni pl:)*
  $\{t \mid (R(t) \vee S(t)) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos
- $\left\{t^{(2)} \mid \exists u_1 \exists u_2 \left(R(u_1) \wedge R(u_2) \wedge t[1] = u_2[1] \wedge t[2] = u_1[1] \wedge \phi(t, u_1, u_2)\right)\right\}$
- $\exists u(R(u) \wedge \dots)$ ha itt a hátrább levő részek biztonságosak (az R-re való szűkítés miatt (ii) teljesül)

Biztonságos formulák

Tipikus technikák a biztonságosság elérésére:

- $\{t \mid R(t) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos (mert szűkítés R-re, így (i) teljesül)
- *(Ha nem lehet egy relációra korlátozni pl:)*
  $\{t \mid (R(t) \vee S(t)) \wedge \phi(t)\}$ biztonságos, ha ϕ biztonságos
- $\left\{t^{(2)} \mid \exists u_1 \exists u_2 \left(R(u_1) \wedge R(u_2) \wedge t[1] = u_2[1] \wedge t[2] = u_1[1] \wedge \phi(t, u_1, u_2)\right)\right\}$
- $\exists u(R(u) \wedge \dots)$ ha itt a hátrább levő részek biztonságosak (az R-re való szűkítés miatt (ii) teljesül)
- $\forall u(\neg R(u) \vee \dots)$ ha itt a hátrább levő részek biztonságosak (mert ez átírva $\neg\exists u(R(u) \wedge \dots)$

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Bizonyítás.

Ötlet: Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ megengedett reláció sorkalkulusban, akkor $R \cap Dom(\phi)^k$ kifejezhető rel. algebrával.

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Bizonyítás.

Ötlet: Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ megengedett reláció sorkalkulusban, akkor $R \cap Dom(\phi)^k$ kifejezhető rel. algebrával. Biz. ϕ felépítése szerinti indukcióval.

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Bizonyítás.

Ötlet: Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ megengedett reláció sorkalkulusban, akkor $R \cap Dom(\phi)^k$ kifejezhető rel. algebrával. Biz. ϕ felépítése szerinti indukcióval.
Ha R biztonságos, akkor $R = R \cap Dom(\phi)^k$ $\surd$

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Bizonyítás.

Ötlet: Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ megengedett reláció sorkalkulusban, akkor $R \cap Dom(\phi)^k$ kifejezhető rel. algebrával. Biz. ϕ felépítése szerinti indukcióval.
Ha R biztonságos, akkor $R = R \cap Dom(\phi)^k$ $\surd$ □

Tétel

A relációs algebra és a biztonságos sorkalkulus ekvivalensek.

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Bizonyítás.

Ötlet: Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ megengedett reláció sorkalkulusban, akkor $R \cap Dom(\phi)^k$ kifejezhető rel. algebrával. Biz. ϕ felépítése szerinti indukcióval.
Ha R biztonságos, akkor $R = R \cap Dom(\phi)^k$ $\surd$ □

Tétel

A relációs algebra és a biztonságos sorkalkulus ekvivalensek.

Bizonyítás.

Volt: relációs algebrai kifejezésből lehet sorkalkulust csinálni, illetve biztonságos sorkalkulusból relációs algebrát.

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Bizonyítás.

Ötlet: Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ megengedett reláció sorkalkulusban, akkor $R \cap Dom(\phi)^k$ kifejezhető rel. algebrával. Biz. ϕ felépítése szerinti indukcióval.
Ha R biztonságos, akkor $R = R \cap Dom(\phi)^k$ $\surd$ □

Tétel

A relációs algebra és a biztonságos sorkalkulus ekvivalensek.

Bizonyítás.

Volt: relációs algebrai kifejezésből lehet sorkalkulust csinálni, illetve biztonságos sorkalkulusból relációs algebrát.
Kell még: a relációs alg. átírása sorkalkulusra biztonságos.

Biztonságos formulák

Tétel

Biztonságos kifejezésű sorkalkulus reláció kifejezhető relációs algebrával is.

Bizonyítás.

Ötlet: Ha $R = \left\{ t^{(k)} \mid \phi(t) \right\}$ megengedett reláció sorkalkulusban, akkor $R \cap Dom(\phi)^k$ kifejezhető rel. algebrával. Biz. ϕ felépítése szerinti indukcióval.
Ha R biztonságos, akkor $R = R \cap Dom(\phi)^k$ $\surd$ □

Tétel

A relációs algebra és a biztonságos sorkalkulus ekvivalensek.

Bizonyítás.

Volt: relációs algebrai kifejezésből lehet sorkalkulust csinálni, illetve biztonságos sorkalkulusból relációs algebrát.
Kell még: a relációs alg. átírása sorkalkulusra biztonságos.
De az meg az volt, ahogy csináltuk. $\surd$ □

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right. \right.$$
$$\left. \left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \wedge \exists w^{(1)} (w[1] \neq t[1]) \right\}$$

  Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.
  Bizt.:
  $\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \right\}$
  Másik bizt.:
  $\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \neg \exists v^{(2)} \left(\text{BEVÉTEL}(v) \wedge v[2] > u[2] \right) \right) \right\}$
- Mely árukból adtak el 100-nál többet egy napon?
  Nem bizt.:
  $\left\{ t^{(3)} \mid \text{ÁRU}(t) \wedge \exists v^{(3)} \left((v[2] = t[1] \wedge v[3] > 100) \boxed{\vee} \text{MENNYISÉG}(v) \right) \right\}$
  Elírtuk: nem is azt adja, amit kell, meg nem is biztonságos, mert (ii) sérül
  Bizt.:
  $\left\{ t^{(3)} \mid \text{ÁRU}(t) \wedge \exists v^{(3)} \left((v[2] = t[1] \wedge v[3] > 100) \boxed{\wedge} \text{MENNYISÉG}(v) \right) \right\}$

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right. \right.$$
$$\left. \left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \wedge \exists w^{(1)} (w[1] \neq t[1]) \right\}$$

Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right. \right.$$
$$\left. \left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \wedge \exists w^{(1)} (w[1] \neq t[1]) \right\}$$

  Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.
  Bizt.:
  $\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \right\}$

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right. \right.$$
$$\left. \left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \wedge \exists w^{(1)} (w[1] \neq t[1]) \right\}$$

Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.

Bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \right\}$$

Másik bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \neg \exists v^{(2)} \left(\text{BEVÉTEL}(v) \wedge v[2] > u[2] \right) \right) \right\}$$

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right.\right.$$
$$\left.\left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2]\right)\right) \wedge \exists w^{(1)}(w[1] \neq t[1])\right\}$$

  Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.
  Bizt.:
  $\left\{t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2]\right)\right)\right\}$
  Másik bizt.:
  $\left\{t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \neg\exists v^{(2)} \left(\text{BEVÉTEL}(v) \wedge v[2] > u[2]\right)\right)\right\}$
- Mely árukból adtak el 100-nál többet egy napon?

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right.\right.$$
$$\left.\left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2]\right)\right) \wedge \exists w^{(1)}(w[1] \neq t[1])\right\}$$

  Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.
  Bizt.:
  $\left\{t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2]\right)\right)\right\}$
  Másik bizt.:
  $\left\{t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \neg\exists v^{(2)} \left(\text{BEVÉTEL}(v) \wedge v[2] > u[2]\right)\right)\right\}$
- Mely árukból adtak el 100-nál többet egy napon?
  Nem bizt.:
  $\left\{t^{(3)} \mid \text{ÁRU}(t) \wedge \exists v^{(3)} \left((v[2] = t[1] \wedge v[3] > 100) \boxed{\vee} \text{MENNYISÉG}(v)\right)\right\}$

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right. \right.$$
$$\left. \left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \wedge \exists w^{(1)} (w[1] \neq t[1]) \right\}$$

  Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.
  Bizt.:
  $\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \right\}$
  Másik bizt.:
  $\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \neg \exists v^{(2)} \left(\text{BEVÉTEL}(v) \wedge v[2] > u[2] \right) \right) \right\}$
- Mely árukból adtak el 100-nál többet egy napon?
  Nem bizt.:
  $\left\{ t^{(3)} \mid \text{ÁRU}(t) \wedge \exists v^{(3)} \left((v[2] = t[1] \wedge v[3] > 100) \boxed{\vee} \text{MENNYISÉG}(v) \right) \right\}$
  Elírtuk: nem is azt adja, amit kell, meg nem is biztonságos, mert (ii) sérül

Példák biztonságos és nem biztonságos kifejezésekre

- Melyik napokon volt a legnagyobb a bevétel?
  Nem bizt.:

$$\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \right. \right.$$
$$\left. \left. \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \wedge \exists w^{(1)} (w[1] \neq t[1]) \right\}$$

  Ez azt fejezi ki, csak nem biztonságos, mert (ii) nem oké, bár (i) az. Az a baj, hogy a végén van egy felesleges dolog.
  Bizt.:
  $\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \forall v^{(2)} \left(\neg \text{BEVÉTEL}(v) \vee v[2] \leq u[2] \right) \right) \right\}$
  Másik bizt.:
  $\left\{ t^{(1)} \mid \exists u^{(2)} \left(\text{BEVÉTEL}(u) \wedge t[1] = u[1] \wedge \neg \exists v^{(2)} \left(\text{BEVÉTEL}(v) \wedge v[2] > u[2] \right) \right) \right\}$
- Mely árukból adtak el 100-nál többet egy napon?
  Nem bizt.:
  $\left\{ t^{(3)} \mid \text{ÁRU}(t) \wedge \exists v^{(3)} \left((v[2] = t[1] \wedge v[3] > 100) \boxed{\vee} \text{MENNYISÉG}(v) \right) \right\}$
  Elírtuk: nem is azt adja, amit kell, meg nem is biztonságos, mert (ii) sérül
  Bizt.:
  $\left\{ t^{(3)} \mid \text{ÁRU}(t) \wedge \exists v^{(3)} \left((v[2] = t[1] \wedge v[3] > 100) \boxed{\wedge} \text{MENNYISÉG}(v) \right) \right\}$

Adatbázisok elmélete

Oszlopkalkulus, más lekérdezőnyelvek

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

8. előadás

Oszlopkalkulus (Domain calculus)

Lényegileg ugyanaz, mint a sorkalkulus, csak másféle változókat használ.
Oszlopváltozó: egy koordinátája a sorváltozónak, mint vektornak
Az oszlopváltozó értékkészlete: egy adott attribútum értékkészlete.

Oszlopkalkulus (Domain calculus)

Lényegileg ugyanaz, mint a sorkalkulus, csak másféle változókat használ.
Oszlopváltozó: egy koordinátája a sorváltozónak, mint vektornak
Az oszlopváltozó értékkészlete: egy adott attribútum értékkészlete.
A oszlopkalkulus által kifejezett reláció:
$\{u_1, \ldots, u_k \mid \phi(u_1, \ldots, u_k)\} \implies$ azon $u = (u_1, \ldots, u_k)$ vektorok, amikre $\phi(u_1, \ldots, u_k)$ igaz, ahol ϕ egy megengedett formula és csak $u_1, \ldots, u_k$ szabad változók benne.

Oszlopkalkulus (Domain calculus)

Lényegileg ugyanaz, mint a sorkalkulus, csak másféle változókat használ.
Oszlopváltozó: egy koordinátája a sorváltozónak, mint vektornak
Az oszlopváltozó értékkészlete: egy adott attribútum értékkészlete.
A oszlopkalkulus által kifejezett reláció:
$\{u_1, \ldots, u_k \mid \phi(u_1, \ldots, u_k)\} \implies$ azon $u = (u_1, \ldots, u_k)$ vektorok, amikre $\phi(u_1, \ldots, u_k)$ igaz, ahol ϕ egy megengedett formula és csak $u_1, \ldots, u_k$ szabad változók benne.

Oszlopkalkulus

Megengedett formulák:

atomok :

- $R^{(k)}(u_1, \ldots, u_k)$: *akkor igaz, ha* $(u_1, \ldots, u_k) \in R$, *azaz a sor benne van a relációban.*

Oszlopkalkulus

Megengedett formulák:

atomok :

- $R^{(k)}(u_1, \ldots, u_k)$: *akkor igaz, ha* $(u_1, \ldots, u_k) \in R$, *azaz a sor benne van a relációban.*
-
  - $u_i \ \theta \ u_j$
  - $u_i \ \theta \ c$
  - $c \ \theta \ u_i$

  ahol $\theta \in \{<, >, =, \neq, \leq, \geq\}$, u_i oszlopváltozók, c konstans érték. *Világos, mikor igaz.*

építkezési szabályok :

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák. *Világos, hogy mikor igaz.*

Oszlopkalkulus

Megengedett formulák:

atomok :

- $R^{(k)}(u_1, \ldots, u_k)$: *akkor igaz, ha* $(u_1, \ldots, u_k) \in R$, *azaz a sor benne van a relációban.*
-
  - $u_i \ \theta \ u_j$
  - $u_i \ \theta \ c$
  - $c \ \theta \ u_i$

  ahol $\theta \in \{<, >, =, \neq, \leq, \geq\}$, u_i oszlopváltozók, c konstans érték. *Világos, mikor igaz.*

építkezési szabályok :

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák. *Világos, hogy mikor igaz.*
- ϕ formula, s oszlopváltozó, akkor $\forall s\phi, \exists s\phi$ is formula. *Világos, hogy mikor igaz.*

Oszlopkalkulus

Megengedett formulák:

atomok :

- $R^{(k)}(u_1, \ldots, u_k)$: *akkor igaz, ha* $(u_1, \ldots, u_k) \in R$, *azaz a sor benne van a relációban.*
-
  - $u_i \ \theta \ u_j$
  - $u_i \ \theta \ c$
  - $c \ \theta \ u_i$

  ahol $\theta \in \{<, >, =, \neq, \leq, \geq\}$, u_i oszlopváltozók, c konstans érték. *Világos, mikor igaz.*

építkezési szabályok :

- ϕ, ψ formulák, akkor $\phi \vee \psi, \phi \wedge \psi, \neg\phi$ is formulák. *Világos, hogy mikor igaz.*
- ϕ formula, s oszlopváltozó, akkor $\forall s\phi, \exists s\phi$ is formula. *Világos, hogy mikor igaz.*
  Kötött változó: ha vonatkozik rá kvantor,
  Szabad változó: ha nem,

Példák oszlopkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Példák oszlopkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG$-$4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt, legyen előbb a dátum:

$$\left\{ t^{(2)} \mid \text{BEVÉTEL}(t) \wedge t[1] \geq \text{2004-01-01} \right\}$$

Példák oszlopkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt, legyen előbb a dátum:

$$\left\{ t^{(2)} \mid \text{BEVÉTEL}(t) \wedge t[1] \geq \text{2004-01-01} \right\}$$

$$\left\{ y, x \mid \text{BEVÉTEL}(y, x) \wedge y \geq \text{2004-01-01} \right\}$$

Az 2004. jan. 15-i bevétel és a befizetett összeg:

$$\left\{ u^{(2)} \mid \text{BEFIZ}(u) \wedge \exists v(\text{BEVÉTEL}(v) \wedge v[1] = \text{2004-01-15} \wedge v[2] = u[1]) \right\}$$

Példák oszlopkalkulus alkalmazására

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)
BEVÉTEL(DÁTUM, ÖSSZEG)
BEFIZ(ÖSSZEG, BEFIZ) BEFIZ=ÖSSZEG−4000

Az 2004. jan. 1. utáni napok bevételei a dátummal együtt, legyen előbb a dátum:

$$\left\{ t^{(2)} \mid \text{BEVÉTEL}(t) \wedge t[1] \geq \text{2004-01-01} \right\}$$

$$\left\{ y, x \mid \text{BEVÉTEL}(y, x) \wedge y \geq \text{2004-01-01} \right\}$$

Az 2004. jan. 15-i bevétel és a befizetett összeg:

$$\left\{ u^{(2)} \mid \text{BEFIZ}(u) \wedge \exists v(\text{BEVÉTEL}(v) \wedge v[1] = \text{2004-01-15} \wedge v[2] = u[1]) \right\}$$

$$\left\{ x, y \mid \text{BEFIZ}(x, y) \wedge \exists z(\text{BEVÉTEL}(z, x) \wedge z = \text{2004-01-15}) \right\}$$

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\Big\{ s^{(3)} \mid \exists u \exists v \Big(\text{MENNYISÉG}(u) \wedge \text{ÁRU}(v) \wedge u[1] = \text{2004-01-15} \wedge u[2] = \text{'A123'} \wedge$$
$$v[1] = \text{'A123'} \wedge s[1] = u[3] \wedge s[2] = v[2] \wedge s[3] = v[3] \Big) \Big\}$$

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\left\{ s^{(3)} \mid \exists u \exists v \Big(\text{MENNYISÉG}(u) \wedge \text{ÁRU}(v) \wedge u[1] = \text{2004-01-15} \wedge u[2] = \text{'A123'} \wedge \right.$$
$$\left. v[1] = \text{'A123'} \wedge s[1] = u[3] \wedge s[2] = v[2] \wedge s[3] = v[3] \Big) \right\}$$

$$\left\{ x, y, z \mid \exists u \exists v \Big(\text{MENNYISÉG}(u, v, x) \wedge \text{ÁRU}(v, y, z) \wedge u = \text{2004-01-15} \wedge v = \text{'A123'} \Big) \right\}$$

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\left\{ s^{(3)} \mid \exists u \exists v \Big(\text{MENNYISÉG}(u) \wedge \text{ÁRU}(v) \wedge u[1] = \text{2004-01-15} \wedge u[2] = \text{'A123'} \wedge v[1] = \text{'A123'} \wedge s[1] = u[3] \wedge s[2] = v[2] \wedge s[3] = v[3] \Big) \right\}$$

$$\left\{ x, y, z \mid \exists u \exists v \Big(\text{MENNYISÉG}(u, v, x) \wedge \text{ÁRU}(v, y, z) \wedge u = \text{2004-01-15} \wedge v = \text{'A123'} \Big) \right\}$$

Mely nevű áruk azok, amelyekkel van azonos egységárú másik áru?

$$\left\{ s^{(1)} \mid \exists u \exists v \Big(\text{ÁRU}(v) \wedge \text{ÁRU}(u) \wedge s[1] = v[2] \wedge v[3] = u[3] \wedge \neg(v[1] = u[1]) \Big) \right\}$$

Hány darabot adtak el 2004. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\left\{ s^{(3)} \mid \exists u \exists v \Big(\text{MENNYISÉG}(u) \wedge \text{ÁRU}(v) \wedge u[1] = \text{2004-01-15} \wedge u[2] = \text{'A123'} \wedge v[1] = \text{'A123'} \wedge s[1] = u[3] \wedge s[2] = v[2] \wedge s[3] = v[3] \Big) \right\}$$

$$\left\{ x, y, z \mid \exists u \exists v \Big(\text{MENNYISÉG}(u, v, x) \wedge \text{ÁRU}(v, y, z) \wedge u = \text{2004-01-15} \wedge v = \text{'A123'} \Big) \right\}$$

Mely nevű áruk azok, amelyekkel van azonos egységárú másik áru?

$$\left\{ s^{(1)} \mid \exists u \exists v \Big(\text{ÁRU}(v) \wedge \text{ÁRU}(u) \wedge s[1] = v[2] \wedge v[3] = u[3] \wedge \neg (v[1] = u[1]) \Big) \right\}$$

$$\left\{ v \mid \exists x \exists y \exists w \exists u \Big(\text{ÁRU}(x, v, u) \wedge \text{ÁRU}(y, w, u) \wedge x \neq y \Big) \right\}$$

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Tétel

A sorkalkulus és az oszlopkalkulus ekvivalensek. A biztonságos sorkalkulus és a biztonságos oszlopkalkulus ekvivalensek.

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Tétel

A sorkalkulus és az oszlopkalkulus ekvivalensek. A biztonságos sorkalkulus és a biztonságos oszlopkalkulus ekvivalensek.

Bizonyítás.

Vázlat:

$$\left\{t^{(k)} \mid \phi\left(t^{(k)}\right)\right\} \quad \longleftrightarrow \quad \{u_1, \ldots, u_k \mid \psi(u_1, \ldots, u_k)\}$$

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Tétel

A sorkalkulus és az oszlopkalkulus ekvivalensek. A biztonságos sorkalkulus és a biztonságos oszlopkalkulus ekvivalensek.

Bizonyítás.

Vázlat:

$$\left\{t^{(k)} \mid \phi\left(t^{(k)}\right)\right\} \quad \longleftrightarrow \quad \{u_1, \ldots, u_k \mid \psi(u_1, \ldots, u_k)\}$$

$$t^{(k)} \quad \longleftrightarrow \quad u_1, \ldots, u_k$$

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Tétel

A sorkalkulus és az oszlopkalkulus ekvivalensek. A biztonságos sorkalkulus és a biztonságos oszlopkalkulus ekvivalensek.

Bizonyítás.

Vázlat:

$$\left\{t^{(k)} \mid \phi\left(t^{(k)}\right)\right\} \quad \longleftrightarrow \quad \{u_1, \ldots, u_k \mid \psi(u_1, \ldots, u_k)\}$$

$$t^{(k)} \quad \longleftrightarrow \quad u_1, \ldots, u_k$$

$$R\left(t^{(k)}\right) \quad \longleftrightarrow \quad R(u_1, \ldots, u_k)$$

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Tétel

A sorkalkulus és az oszlopkalkulus ekvivalensek. A biztonságos sorkalkulus és a biztonságos oszlopkalkulus ekvivalensek.

Bizonyítás.

Vázlat:

$$\begin{aligned}
\left\{ t^{(k)} \mid \phi\left(t^{(k)}\right) \right\} &\longleftrightarrow \{u_1, \ldots, u_k \mid \psi(u_1, \ldots, u_k)\} \\
t^{(k)} &\longleftrightarrow u_1, \ldots, u_k \\
R\left(t^{(k)}\right) &\longleftrightarrow R(u_1, \ldots, u_k) \\
t^{(k)}[j] &\longleftrightarrow u_j
\end{aligned}$$

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.

Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Tétel

A sorkalkulus és az oszlopkalkulus ekvivalensek. A biztonságos sorkalkulus és a biztonságos oszlopkalkulus ekvivalensek.

Bizonyítás.

Vázlat:

$$\begin{aligned}
\left\{t^{(k)} \mid \phi\left(t^{(k)}\right)\right\} &\longleftrightarrow \{u_1,\ldots,u_k \mid \psi(u_1,\ldots,u_k)\}\\
t^{(k)} &\longleftrightarrow u_1,\ldots,u_k\\
R\left(t^{(k)}\right) &\longleftrightarrow R(u_1,\ldots,u_k)\\
t^{(k)}[j] &\longleftrightarrow u_j\\
\exists t^{(k)}\phi\left(t^{(k)}\right) &\longleftrightarrow \exists u_1 \ldots \exists u_k \psi(u_1,\ldots,u_k)
\end{aligned}$$

Biztonságos kifejezés

Dom(ψ) és a biztonságos formula és kifejezés ugyanaz, mint sorkalkulusnál.
Ugyanolyan technikák vannak a biztonságosság elérésére, mint sorkalkulusnál.

Tétel

A sorkalkulus és az oszlopkalkulus ekvivalensek. A biztonságos sorkalkulus és a biztonságos oszlopkalkulus ekvivalensek.

Bizonyítás.

Vázlat:

$$\begin{aligned}
\left\{t^{(k)} \mid \phi\left(t^{(k)}\right)\right\} &\longleftrightarrow \{u_1,\ldots,u_k \mid \psi(u_1,\ldots,u_k)\}\\
t^{(k)} &\longleftrightarrow u_1,\ldots,u_k\\
R\left(t^{(k)}\right) &\longleftrightarrow R(u_1,\ldots,u_k)\\
t^{(k)}[j] &\longleftrightarrow u_j\\
\exists t^{(k)}\phi\left(t^{(k)}\right) &\longleftrightarrow \exists u_1\ldots\exists u_k\psi(u_1,\ldots,u_k)\\
\text{biztonságos} &\longleftrightarrow \text{biztonságos}
\end{aligned}$$

□

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Általában a konkrét lekérdezőnyelvek eltérnek a modelltől (algebrai és logikai esetben is), van amiben többet tudnak, van amiben kevesebbet, vagy csak máshogy.

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Általában a konkrét lekérdezőnyelvek eltérnek a modelltől (algebrai és logikai esetben is), van amiben többet tudnak, van amiben kevesebbet, vagy csak máshogy.
Lehetséges eltérések:

- logikai alapúakban eleve csak biztonságos kifejezéseknek megfelelő kérdéseket lehet írni (nincsenek kvantorok)

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Általában a konkrét lekérdezőnyelvek eltérnek a modelltől (algebrai és logikai esetben is), van amiben többet tudnak, van amiben kevesebbet, vagy csak máshogy.
Lehetséges eltérések:

- logikai alapúakban eleve csak biztonságos kifejezéseknek megfelelő kérdéseket lehet írni (nincsenek kvantorok)
- aritmetika

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Általában a konkrét lekérdezőnyelvek eltérnek a modelltől (algebrai és logikai esetben is), van amiben többet tudnak, van amiben kevesebbet, vagy csak máshogy.
Lehetséges eltérések:

- logikai alapúakban eleve csak biztonságos kifejezéseknek megfelelő kérdéseket lehet írni (nincsenek kvantorok)
- aritmetika
- aggregátumok

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Általában a konkrét lekérdezőnyelvek eltérnek a modelltől (algebrai és logikai esetben is), van amiben többet tudnak, van amiben kevesebbet, vagy csak máshogy.
Lehetséges eltérések:

- logikai alapúakban eleve csak biztonságos kifejezéseknek megfelelő kérdéseket lehet írni (nincsenek kvantorok)
- aritmetika
- aggregátumok
- többszörös sorok kezelése/megengedése

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Általában a konkrét lekérdezőnyelvek eltérnek a modelltől (algebrai és logikai esetben is), van amiben többet tudnak, van amiben kevesebbet, vagy csak máshogy.
Lehetséges eltérések:

- logikai alapúakban eleve csak biztonságos kifejezéseknek megfelelő kérdéseket lehet írni (nincsenek kvantorok)
- aritmetika
- aggregátumok
- többszörös sorok kezelése/megengedése
- attribútumok sorrendje kötött

Lekérdezőnyelvek típusai, általános jellemzőik

- Lehetnek algebrai alapúak: relációs algebrán alapuló lekérdezés, procedurális leírás
  Előny: lekérdezésoptimalizálásra jobb
- Lehetnek logikai alapúak: sor vagy oszlopkalkulusra épülő lekérdezés, deklaratív leírás
  Előny: könnyebben átlátható

Általában a konkrét lekérdezőnyelvek eltérnek a modelltől (algebrai és logikai esetben is), van amiben többet tudnak, van amiben kevesebbet, vagy csak máshogy.
Lehetséges eltérések:

- logikai alapúakban eleve csak biztonságos kifejezéseknek megfelelő kérdéseket lehet írni (nincsenek kvantorok)
- aritmetika
- aggregátumok
- többszörös sorok kezelése/megengedése
- attribútumok sorrendje kötött

De az igaz mindre, hogy relációsan teljesek (esetleg bizonyos műveletek nehezebben mennek) és általában van a lekérdező funkció mellett DML és DDL is.

Példák relációs adatbáziskezelő nyelvekre

- Information System Base Language $\Longrightarrow$ ISBL
  - relációs algebra alapú
  - kifejlesztő: IBM's United Kingdom Scientific Center
  - Peterlee Relational Test Vehicle

Példák relációs adatbáziskezelő nyelvekre

- Information System Base Language $\Longrightarrow$ ISBL
  - relációs algebra alapú
  - kifejlesztő: IBM's United Kingdom Scientific Center
  - Peterlee Relational Test Vehicle
- QUEry Language $\Longrightarrow$ QUEL
  - sorkalkulus alapú
  - kifejlesztő: University of California at Berkly
  - INGRES lekérdezőnyelve

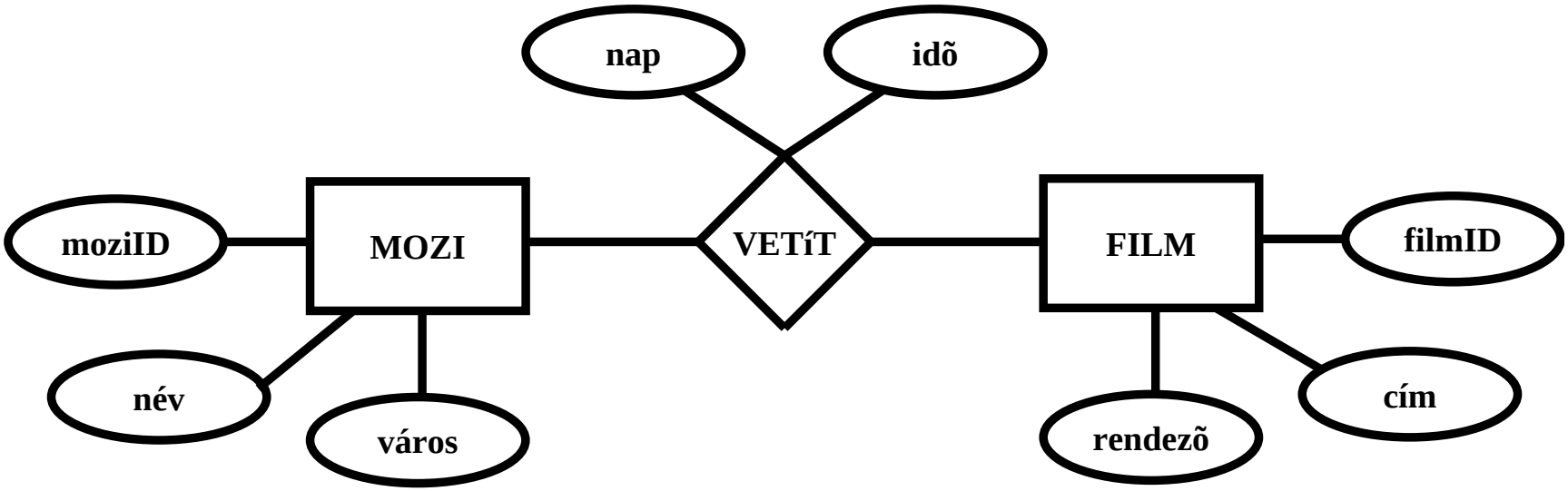
Példák relációs adatbáziskezelő nyelvekre

- Information System Base Language $\Longrightarrow$ ISBL
  - relációs algebra alapú
  - kifejlesztő: IBM's United Kingdom Scientific Center
  - Peterlee Relational Test Vehicle
- QUEry Language $\Longrightarrow$ QUEL
  - sorkalkulus alapú
  - kifejlesztő: University of California at Berkly
  - INGRES lekérdezőnyelve
- Query-By-Example $\Longrightarrow$ QBE
  - oszlopkalkulus alapú
  - kifejlesztő: IBM's Watson Research Center
  - pl. MS Access

Példák relációs adatbáziskezelő nyelvekre

- Information System Base Language $\Longrightarrow$ ISBL
  - relációs algebra alapú
  - kifejlesztő: IBM's United Kingdom Scientific Center
  - Peterlee Relational Test Vehicle
- QUEry Language $\Longrightarrow$ QUEL
  - sorkalkulus alapú
  - kifejlesztő: University of California at Berkly
  - INGRES lekérdezőnyelve
- Query-By-Example $\Longrightarrow$ QBE
  - oszlopkalkulus alapú
  - kifejlesztő: IBM's Watson Research Center
  - pl. MS Access
- Structured Query Language $\Longrightarrow$ SQL
  - oszlopkalkulus-szerű alapjai vannak, némi sorkalkulus elemmel
  - kifejlesztő: IBM's San Jose Research Laboratory
  - pl. Oracle, MS SQL, IBM DB2, MySQL, PostgreSQL

A példákban használt relációs séma



MOZI	moziID	név	város
	101	Művész	Budapest
	102	Uránia	Pécs
	⋮		

FILM	filmID	cím	rendező
	1	Macskajaj	E. Kusturica
	2	Moszkva tér	Török F.
	⋮		

VETÍT	moziID	filmID	nap	idő
	101	1	péntek	16:00
	101	2	szombat	19:00
	⋮			

ISBL

- Relációs algebra alapú lekérdezések
  - operátorok

művelet	relációs algebra	ISBL
unió	$R \cup S$	**R + S**
metszet	$R \cap S$	**R . S**
természetes illesztés	$R \bowtie S$	**R * S**
különbség (általánosabb)	$R \setminus (R \overline{\bowtie} S)$	**R – S**
kiválasztás	$\sigma_F(R)$	**R:F**
vetítés	$\pi_{A,B,C}(R)$	**R % A,B,C**

ISBL

- Relációs algebra alapú lekérdezések
  - operátorok

művelet	relációs algebra	ISBL
unió	$R \cup S$	**R + S**
metszet	$R \cap S$	**R . S**
természetes illesztés	$R \bowtie S$	**R * S**
különbség (általánosabb)	$R \setminus (R \overline{\bowtie} S)$	**R – S**
kiválasztás	$\sigma_F(R)$	**R:F**
vetítés	$\pi_{A,B,C}(R)$	**R % A,B,C**

  - egyéb elemek
    - ★ eredmény megjelenítése: **list** kulcsszó
    - ★ értékadás relációnak: **=**
    - ★ és, vagy, tagadás jelei: **&, |, ¬**

ISBL

- Relációs algebra alapú lekérdezések
  - operátorok

művelet	relációs algebra	ISBL
unió	$R \cup S$	**R + S**
metszet	$R \cap S$	**R . S**
természetes illesztés	$R \bowtie S$	**R * S**
különbség (általánosabb)	$R \setminus (R \overline{\bowtie} S)$	**R – S**
kiválasztás	$\sigma_F(R)$	**R:F**
vetítés	$\pi_{A,B,C}(R)$	**R % A,B,C**

  - egyéb elemek
    - ★ eredmény megjelenítése: **list** kulcsszó
    - ★ értékadás relációnak: **=**
    - ★ és, vagy, tagadás jelei: **&, |,** $\neg$
- További nyelvi elemek
  - aggregátumok (min., max., összeg, átlag, darabszám) kezelése
  - adatfrissítő műveletek
  - átnevezés
  - kimenet formázása

ISBL példák

- Budapesti mozik nevei

  list MOZI **:** város = "Budapest" **%** név

ISBL példák

- Budapesti mozik nevei

  list MOZI **:** város = "Budapest" **%** név

- Pénteken 16 órakor kezdődő filmek címe, rendezője

  list FILM ***** VETÍT **:** nap = "péntek" & idő = "16:00" **%** cím, rendező

ISBL példák

- Budapesti mozik nevei

  list MOZI **:** város = "Budapest" **%** név

- Pénteken 16 órakor kezdődő filmek címe, rendezője

  list FILM * VETÍT **:** nap = "péntek" & idő = "16:00" **%** cím, rendező

- Pénteken nem vetített filmek címe

  list FILM **–** (VETÍT **:** nap = "péntek") **%** cím

  (Figyelem! Különbségnek más a definíciója.)

ISBL példák

- Budapesti mozik nevei

  list MOZI **:** város = "Budapest" **%** név

- Pénteken 16 órakor kezdődő filmek címe, rendezője

  list FILM * VETÍT **:** nap = "péntek" & idő = "16:00" **%** cím, rendező

- Pénteken nem vetített filmek címe

  list FILM **–** (VETÍT **:** nap = "péntek") **%** cím

  (Figyelem! Különbségnek más a definíciója.)

- Pénteken és szombaton is vetített filmek címe

  list (FILM * VETÍT **:** nap = "péntek" **%** cím) .
  (FILM * VETÍT **:** nap = "szombat" **%** cím)

ISBL példák

- Budapesti mozik nevei

  list MOZI **:** város = "Budapest" **%** név

- Pénteken 16 órakor kezdődő filmek címe, rendezője

  list FILM * VETÍT **:** nap = "péntek" & idő = "16:00" **%** cím, rendező

- Pénteken nem vetített filmek címe

  list FILM **–** (VETÍT **:** nap = "péntek") **%** cím

  (Figyelem! Különbségnek más a definíciója.)

- Pénteken és szombaton is vetített filmek címe

  list (FILM * VETÍT **:** nap = "péntek" **%** cím) .
  (FILM * VETÍT **:** nap = "szombat" **%** cím)

  vagy köztes relációk bevezetésével:

  r1 **=** FILM * VETÍT **:** nap = "péntek" **%** cím
  r2 **=** FILM * VETÍT **:** nap = "szombat" **%** cím
  list r1.r2

QUEL

Sorkalkulus alapú.
Pl. Budapesti mozik nevei:
$\{t[2] \mid \text{MOZI}(t) \wedge t[3] = \text{'Budapest'}\}$

QUEL

Sorkalkulus alapú.
Pl. Budapesti mozik nevei:
$\{t[2] \mid \text{MOZI}(t) \wedge t[3] = \text{'Budapest'}\}$
Meg kell mondani, hogy a sorváltozó melyik reláció sorain fut:

$$R(t) \quad \longleftrightarrow \quad \textbf{range of } t \textbf{ is } R$$

QUEL

Sorkalkulus alapú.
Pl. Budapesti mozik nevei:
$\{t[2] \mid \text{MOZI}(t) \wedge t[3] = \text{'Budapest'}\}$
Meg kell mondani, hogy a sorváltozó melyik reláció sorain fut:

$$R(t) \quad \longleftrightarrow \quad \textbf{range of } t \textbf{ is } R$$

Hivatkozni kell a sorváltozó komponenseire:

$$t[i] \quad \longleftrightarrow \quad \textbf{t.<}i\textbf{-edik attributum neve>}$$

QUEL

Sorkalkulus alapú.
Pl. Budapesti mozik nevei:
$\{t[2] \mid \text{MOZI}(t) \wedge t[3] = \text{'Budapest'}\}$
Meg kell mondani, hogy a sorváltozó melyik reláció sorain fut:

$$R(t) \quad \longleftrightarrow \quad \textbf{range of } t \textbf{ is } R$$

Hivatkozni kell a sorváltozó komponenseire:

$$t[i] \quad \longleftrightarrow \quad \textbf{t.<}i\textbf{-edik attributum neve>}$$

Pl. $t[3] \longleftrightarrow t$.város

QUEL

Sorkalkulus alapú.
Pl. Budapesti mozik nevei:
$\{t[2] \mid \text{MOZI}(t) \wedge t[3] = \text{'Budapest'}\}$
Meg kell mondani, hogy a sorváltozó melyik reláció sorain fut:

$$R(t) \quad \longleftrightarrow \quad \textbf{range of } t \textbf{ is } R$$

Hivatkozni kell a sorváltozó komponenseire:

$$t[i] \quad \longleftrightarrow \quad \textbf{t.<}i\textbf{-edik attributum neve>}$$

Pl. $t[3] \longleftrightarrow t$.város
Lekérdezés:

retrieve (<lekérdezendő attribútumok> **) where** <feltétel>

QUEL példák

range of *m* is MOZI
range of *f* is FILM
range of *v* is VETÍT
range of *u* is VETÍT

QUEL példák

range of *m* **is MOZI**
range of *f* **is FILM**
range of *v* **is VETÍT**
range of *u* **is VETÍT**

- Budapesti mozik nevei

  retrieve (*m*.név **) where** *m*.város =’Budapest’

QUEL példák

range of *m* **is MOZI**
range of *f* **is FILM**
range of *v* **is VETÍT**
range of *u* **is VETÍT**

- Budapesti mozik nevei

  retrieve (*m*.név **) where** *m*.város =’Budapest’

- Pénteken 16 órakor kezdődő filmek címe, rendezője

  retrieve unique (*f*.cím, *f*.rendező **)**
  where *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’ **and** *v*.idő = ’16:00’

QUEL példák

range of *m* **is MOZI**
range of *f* **is FILM**
range of *v* **is VETÍT**
range of *u* **is VETÍT**

- Budapesti mozik nevei
  retrieve (*m*.név **) where** *m*.város =’Budapest’
- Pénteken 16 órakor kezdődő filmek címe, rendezője
  retrieve unique (*f*.cím, *f*.rendező **)**
  where *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’ **and** *v*.idő = ’16:00’
- Pénteken nem vetített filmek címe:
  $\left\{ t^{(3)}[2] \mid \text{FILM}(t) \wedge \nexists s \left(\text{VETÍT}(s) \wedge s[2] = t[1] \wedge s[3] = \text{’péntek’} \right) \right\}$

QUEL példák

range of *m* **is MOZI**
range of *f* **is FILM**
range of *v* **is VETÍT**
range of *u* **is VETÍT**

- Budapesti mozik nevei
  retrieve (*m*.név **) where** *m*.város =’Budapest’
- Pénteken 16 órakor kezdődő filmek címe, rendezője
  retrieve unique (*f*.cím, *f*.rendező **)**
  where *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’ **and** *v*.idő = ’16:00’
- Pénteken nem vetített filmek címe:
  $\left\{ t^{(3)}[2] \mid \mathrm{FILM}(t) \wedge \nexists s \left(\mathrm{VETÍT}(s) \wedge s[2] = t[1] \wedge s[3] = \text{’péntek’} \right) \right\}$
  QUEL-ben nincsenek igazi kvantorok, csak egy **any** nevű aggregátum!

QUEL példák

range of *m* **is MOZI**
range of *f* **is FILM**
range of *v* **is VETÍT**
range of *u* **is VETÍT**

- Budapesti mozik nevei
  retrieve (*m*.név **) where** *m*.város =’Budapest’
- Pénteken 16 órakor kezdődő filmek címe, rendezője
  retrieve unique (*f*.cím, *f*.rendező **)**
  where *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’ **and** *v*.idő = ’16:00’
- Pénteken nem vetített filmek címe:
  $\left\{ t^{(3)}[2] \mid \text{FILM}(t) \wedge \nexists s \left(\text{VETÍT}(s) \wedge s[2] = t[1] \wedge s[3] = \text{’péntek’} \right) \right\}$
  QUEL-ben nincsenek igazi kvantorok, csak egy **any** nevű aggregátum!
  any()=1, ha van olyan sor, ami kielégíti a feltételt és =0, ha nincs

QUEL példák

range of *m* **is MOZI**
range of *f* **is FILM**
range of *v* **is VETÍT**
range of *u* **is VETÍT**

- Budapesti mozik nevei

  retrieve (*m*.név **) where** *m*.város =’Budapest’
- Pénteken 16 órakor kezdődő filmek címe, rendezője

  retrieve unique (*f*.cím, *f*.rendező **)**
  where *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’ **and** *v*.idő = ’16:00’
- Pénteken nem vetített filmek címe:
  $\left\{ t^{(3)}[2] \mid \mathrm{FILM}(t) \wedge \nexists s \left(\mathrm{VETÍT}(s) \wedge s[2] = t[1] \wedge s[3] = \text{’péntek’} \right) \right\}$
  QUEL-ben nincsenek igazi kvantorok, csak egy **any** nevű aggregátum!
  any()=1, ha van olyan sor, ami kielégíti a feltételt és =0, ha nincs

  retrieve (*f*.cím **)**
  where any (*f*.filmID **where** *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’**)**=0
- Pénteken és szombaton is vetített filmek címe

  retrieve unique (*f*.cím **)**
  where *f*.filmID=*v*.filmID **and** *v*1.filmID=*v*.filmID **and**
  v.nap = ’péntek’ **and** *v*1.nap = ’szombat’

QUEL példák

range of *m* **is MOZI**
range of *f* **is FILM**
range of *v* **is VETÍT**
range of *u* **is VETÍT**

- Budapesti mozik nevei

  retrieve (*m*.név **) where** *m*.város =’Budapest’
- Pénteken 16 órakor kezdődő filmek címe, rendezője

  retrieve unique (*f*.cím, *f*.rendező **)**
  where *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’ **and** *v*.idő = ’16:00’
- Pénteken nem vetített filmek címe:
  $\left\{ t^{(3)}[2] \mid \mathrm{FILM}(t) \wedge \nexists s \left(\mathrm{VETÍT}(s) \wedge s[2] = t[1] \wedge s[3] = \text{’péntek’} \right) \right\}$
  QUEL-ben nincsenek igazi kvantorok, csak egy **any** nevű aggregátum!
  any()=1, ha van olyan sor, ami kielégíti a feltételt és =0, ha nincs

  retrieve (*f*.cím **)**
  where any (*f*.filmID **where** *f*.filmID=*v*.filmID **and** *v*.nap = ’péntek’**)**=0
- Pénteken és szombaton is vetített filmek címe

  retrieve unique (*f*.cím **)**
  where *f*.filmID=*v*.filmID **and** *v*1.filmID=*v*.filmID **and**
  v.nap = ’péntek’ **and** *v*1.nap = ’szombat’

QUEL

Lehet beszúrni és törölni:
retrieve into FILM1 **unique** *f*.cím
retrieve into FILM2 **unique** *f*.cím **where** *f*.filmID=*v*.filmID **and** *v*.nap='péntek'
range of *f*1 **is** FILM1
range of *f*2 **is** FILM2
delete *f*1 **where** *f*1.filmID=*f*2.filmID
retrieve FILM1

QBE

- Oszlopkalkulus alapú lekérdezések, kétdimenziós

QBE

- Oszlopkalkulus alapú lekérdezések, kétdimenziós
  - A lekérdezés elemei változókkal és konstansokkal kitöltött sablon(ok)
    - ★ jelölések

változó	aláhúzott változónév
konstans	nem aláhúzott érték
egyszer említett változó	üres cella
kimenetre kerülő attribútum	**P.** prefix

QBE

- Oszlopkalkulus alapú lekérdezések, kétdimenziós
  - A lekérdezés elemei változókkal és konstansokkal kitöltött sablon(ok)
    - ★ jelölések

változó	aláhúzott változónév
konstans	nem aláhúzott érték
egyszer említett változó	üres cella
kimenetre kerülő attribútum	**P.** prefix

    - ★ Példa: Budapesti mozik nevei:

MOZI	moziID	név	város
	1	**P.**mozinév	Budapest

QBE

- Oszlopkalkulus alapú lekérdezések, kétdimenziós
  - A lekérdezés elemei változókkal és konstansokkal kitöltött sablon(ok)
    - ★ jelölések

változó	aláhúzott változónév
konstans	nem aláhúzott érték
egyszer említett változó	üres cella
kimenetre kerülő attribútum	**P.** prefix

    - ★ Példa: Budapesti mozik nevei:

MOZI	moziID	név	város
	1	**P.**mozinév	Budapest

vagy

MOZI	moziID	név	város
		P.	Budapest

QBE

- Oszlopkalkulus alapú lekérdezések, kétdimenziós
  - A lekérdezés elemei változókkal és konstansokkal kitöltött sablon(ok)
    - ★ jelölések

változó	aláhúzott változónév
konstans	nem aláhúzott érték
egyszer említett változó	üres cella
kimenetre kerülő attribútum	**P.** prefix

    - ★ Példa: Budapesti mozik nevei:

MOZI	moziID	név	város
	1	**P.**mozinév	Budapest

vagy

MOZI	moziID	név	város
		P.	Budapest

  - Összetett lekérdezések is lehetségesek (használatukkor az azonos nevű változók illesztése történik meg).
    - ★ használható több soros sablon (ekkor a kiértékeléskor mindegyik sornak egy-egy futó oszlopváltozó fog megfelelni, és ha illeszkedés van, akkor megtörténik a kiírás)

QBE

- Oszlopkalkulus alapú lekérdezések, kétdimenziós
  - A lekérdezés elemei változókkal és konstansokkal kitöltött sablon(ok)
    - ★ jelölések

változó	aláhúzott változónév
konstans	nem aláhúzott érték
egyszer említett változó	üres cella
kimenetre kerülő attribútum	**P.** prefix

    - ★ Példa: Budapesti mozik nevei:

MOZI	moziID	név	város
	1	**P.**mozinév	Budapest

vagy

MOZI	moziID	név	város
		P.	Budapest

  - Összetett lekérdezések is lehetségesek (használatukkor az azonos nevű változók illesztése történik meg).
    - ★ használható több soros sablon (ekkor a kiértékeléskor mindegyik sornak egy-egy futó oszlopváltozó fog megfelelni, és ha illeszkedés van, akkor megtörténik a kiírás)
    - ★ használható több sablon (kiértékelés hasonlóan, mint a többsoros kérdésnél, csak az oszlopváltozók nem ugyanazon reláció sorait futják be)

- A kiválasztás feltételeinek megadása
  - Az egyenlőség konstanshoz való illesztéssel vizsgálható (mint az előbb),

- A kiválasztás feltételeinek megadása
  - Az egyenlőség konstanshoz való illesztéssel vizsgálható (mint az előbb),
  - egyéb egyszerű relációkhoz a $\neg =, >, <, >=, <, <=$ operátorok használhatók,

- A kiválasztás feltételeinek megadása
  - Az egyenlőség konstanshoz való illesztéssel vizsgálható (mint az előbb),
  - egyéb egyszerű relációkhoz a $\neg =, >, <, >=, <, <=$ operátorok használhatók,
  - összetett feltételeket (pl. két változó közt a $<$ relációt) külön feltételsablon megadásával lehet vizsgálni.

- A kiválasztás feltételeinek megadása
  - Az egyenlőség konstanshoz való illesztéssel vizsgálható (mint az előbb),
  - egyéb egyszerű relációkhoz a $\neg =, >, <, >=, <, <=$ operátorok használhatók,
  - összetett feltételeket (pl. két változó közt a $<$ relációt) külön feltételsablon megadásával lehet vizsgálni.
- További nyelvi elemek
  - mintaillesztés
  - aritmetika
  - kimenet rendezése
  - csoportosítás
  - aggregátumok kezelése
  - reláció tranzitív lezártjának kezelése
  - adatmódosító műveletek
  - típusdefiníció, sémalétrehozás

Megjegyzés: dupla példányt kiirtja, azaz többszörös sorok nincsenek

QBE példák még

- Nem budapesti mozik nevei

MOZI	moziID	név	város
		P.	$\neg =$ Budapest

QBE példák még

- Nem budapesti mozik nevei

MOZI	moziID	név	város
		P.	¬ = Budapest

- Pénteki és szombati kezdési időpontok

VETÍT	moziID	filmID	nap	idő
			P.péntek	**P.**
			P.szombat	**P.**

QBE példák még

- Nem budapesti mozik nevei

MOZI	moziID	név	város
		P.	¬ = Budapest

- Pénteki és szombati kezdési időpontok

VETÍT	moziID	filmID	nap	idő
			P.péntek	**P.**
			P.szombat	**P.**

- Időpontok, amikor pénteken és szombaton is kezdődik film

VETÍT	moziID	filmID	nap	idő
			péntek	**P.**kezdes
			szombat	kezdes

QBE példák még

- Pénteken vetített filmek adatai

FILM	filmID	cím	rendező
	1	P.	P.

VETÍT	moziID	filmID	nap	idő
		1	péntek	

QBE példák még

- Pénteken vetített filmek adatai

FILM	filmID	cím	rendező
	1	P.	P.

VETÍT	moziID	filmID	nap	idő
		1	péntek	

- Azok a városok, ahol van legalább két mozi:

MOZI	moziID	név	város
	1		P.városnév
	2		városnév

CONDITIONS
1≠2

Adatbázisok elmélete

SQL

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

9. előadás

Az SQL nyelv

- Relációs nyelv, mint az eddigiek

Az SQL nyelv

- Relációs nyelv, mint az eddigiek
- oszlopkalkulus jellegű, de némi sorkalkulusos beütéssel

Az SQL nyelv

- Relációs nyelv, mint az eddigiek
- oszlopkalkulus jellegű, de némi sorkalkulusos beütéssel

Termékek, (amik szükségszerűen relációs nyelvet is tartalmaztak):

- IBM: System/R
- Relational Software: Oracle
- Relational Systems: Ingres
- Microsoft: SQL server 2000

Az SQL nyelv

- Relációs nyelv, mint az eddigiek
- oszlopkalkulus jellegű, de némi sorkalkulusos beütéssel

Termékek, (amik szükségszerűen relációs nyelvet is tartalmaztak):

- IBM: System/R
- Relational Software: Oracle
- Relational Systems: Ingres
- Microsoft: SQL server 2000

Szabványok

- SQL89 (SQL1)
- SQL92 (SQL2, mi nagyrészt ezt nézzük most)
- SQL99 (SQL3, ebből is pár dolog, pl. triggerek, rekurzió)

Az SQL nyelv

- Relációs nyelv, mint az eddigiek
- oszlopkalkulus jellegű, de némi sorkalkulusos beütéssel

Termékek, (amik szükségszerűen relációs nyelvet is tartalmaztak):

- IBM: System/R
- Relational Software: Oracle
- Relational Systems: Ingres
- Microsoft: SQL server 2000

Szabványok

- SQL89 (SQL1)
- SQL92 (SQL2, mi nagyrészt ezt nézzük most)
- SQL99 (SQL3, ebből is pár dolog, pl. triggerek, rekurzió)

Működő rendszerekben ezek verziói vannak (főleg SQL2).

Fontosabb utasítások

Data Definition Language:

- CREATE - séma létrehozása
- ALTER - séma módosítása
- DROP - séma törlése

Fontosabb utasítások

Data Definition Language:

- CREATE - séma létrehozása
- ALTER - séma módosítása
- DROP - séma törlése

Data Modification Language:

- INSERT - adatok beszúrása
- UPDATE - adatok módosítása
- DELETE - adatok törlése
- SELECT - adatok lekérdezése

Fontosabb utasítások

Data Definition Language:

- CREATE - séma létrehozása
- ALTER - séma módosítása
- DROP - séma törlése

Data Modification Language:

- INSERT - adatok beszúrása
- UPDATE - adatok módosítása
- DELETE - adatok törlése
- SELECT - adatok lekérdezése

Természetesen előbb mindig a sémát kell létrehozni, és utána dolgozhatunk vele, de most fordítva tárgyaljuk mert eddig a lekérdező nyelvekről volt szó.

DML utasítások — SELECT

Ezzel valósítható meg a kiválasztás, vetítés és a szorzat.
Szintaxis:
SELECT <reláció$_i$>.<attrib$_1$>, . . . , <reláció$_j$>.<attrib$_n$>
FROM <reláció$_1$>, . . . , <reláció$_m$>
WHERE <kifejezés>
Relációs algebrabeli megfelelője (de nem pontosan, mert SQL-ben SELECT nem küszöböli ki a többszörös sorokat):

$$\pi_{\text{<attrib}_1\text{>},\ldots,\text{<attrib}_n\text{>}}\sigma_{\text{<kifejezés>}}(\text{<reláció}_1\text{>} \times \cdots \times \text{<reláció}_m\text{>})$$

Példa 1: A budapesti mozik azonosítói és nevei
SELECT mozi.moziID, mozi.név FROM mozi WHERE mozi.város="Budapest"
Példa 2: A pénteken hétkor kezdődő filmek azonosítói
SELECT vetít.filmID FROM vetít WHERE vetít.nap="péntek" AND vetít.idő="19:00"

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHERE

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHEREfeltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztan arra csinálnánk a kiválasztást és a vetítést.

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHEREfeltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztan arra csinálnánk a kiválasztást és a vetítést.
- ha többszörös sorokat nem akarunk: SELECT DISTINCT (ennek ára van!!!)

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHEREfeltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztan arra csinálnánk a kiválasztást és a vetítést.
- ha többszörös sorokat nem akarunk: SELECT DISTINCT (ennek ára van!!!)
- WHERE el is hagyható

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHEREfeltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztan arra csinálnánk a kiválasztást és a vetítést.
- ha többszörös sorokat nem akarunk: SELECT DISTINCT (ennek ára van!!!)
- WHERE el is hagyható
- WHERE-ben mi állhat: erről később

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHEREfeltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztan arra csinálnánk a kiválasztást és a vetítést.
- ha többszörös sorokat nem akarunk: SELECT DISTINCT (ennek ára van!!!)
- WHERE el is hagyható
- WHERE-ben mi állhat: erről később
- az eredmény az ORDER BY kulcsszó segítségével rendezhető, megadható hogy mely oszlopok szerint és hogy növőleg vagy csökkenőleg

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHEREfeltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztan arra csinálnánk a kiválasztást és a vetítést.
- ha többszörös sorokat nem akarunk: SELECT DISTINCT (ennek ára van!!!)
- WHERE el is hagyható
- WHERE-ben mi állhat: erről később
- az eredmény az ORDER BY kulcsszó segítségével rendezhető, megadható hogy mely oszlopok szerint és hogy növőleg vagy csökkenőleg
- A fenti két példa mutatja, hogy a kiválasztás és a vetítés megy, a szorzatra a sorváltozók bevezetése után nézünk példát

SELECT

Ezzel valósítható meg a kiválasztás, vetítés és a szorzat.
Szintaxis:
SELECT <reláció$_i$>.<attrib$_1$>, . . . , <reláció$_j$>.<attrib$_n$>
FROM <reláció$_1$>, . . . , <reláció$_m$>
WHERE <kifejezés>
Relációs algebrabeli megfelelője (de nem pontosan, mert SQL-ben SELECT nem küszöböli ki a többszörös sorokat):

$$\pi_{\text{<attrib}_1\text{>},\ldots,\text{<attrib}_n\text{>}}\sigma_{\text{<kifejezés>}}(\text{<reláció}_1\text{>} \times \cdots \times \text{<reláció}_m\text{>})$$

Példa 3: A budapesti mozik azonosítói és nevei
SELECT mozi.moziID, mozi.név FROM mozi WHERE mozi.város='Budapest'
Példa 4: A pénteken hétkor kezdődő filmek azonosítói
SELECT vetít.filmID FROM vetít WHERE vetít.nap='péntek' AND vetít.idő='19:00'

Megjegyzés:

- kiértékelés: minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHERE feltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztan arra csinálnánk a kiválasztást és a vetítést.
- ha többszörös sorokat nem akarunk: SELECT DISTINCT (ennek ára van!!!)
- WHERE el is hagyható
- WHERE-ben mi állhat: erről később
- az eredmény az ORDER BY kulcsszó segítségével rendezhető, megadható hogy mely oszlopok szerint és hogy növőleg vagy csökkenőleg
- A fenti két példa mutatja, hogy a kiválasztás és a vetítés megy, a szorzatra a sorváltozók bevezetése után nézünk példát

SQL Sor- és oszlopváltozók

A FROM után felsorolt relációkhoz **sorváltozókat** rendelhetünk.
Szintaxis (FROM után <reláció$_i$> helyén): <reláció$_i$> AS <sorváltozó>

SQL Sor- és oszlopváltozók

A FROM után felsorolt relációkhoz **sorváltozókat** rendelhetünk.
Szintaxis (FROM után <reláció$_i$> helyén): <reláció$_i$> AS <sorváltozó>
A SELECT után elhelyezett attribútum-hivatkozásokhoz **oszlopváltozókat** rendelhetünk.
Szintaxis (SELECT után <reláció$_i$>.<attrib$_j$> helyén):
<reláció$_i$>.<attrib$_j$> AS <oszlopváltozó>

SQL Sor- és oszlopváltozók

A FROM után felsorolt relációkhoz **sorváltozókat** rendelhetünk.
Szintaxis (FROM után <reláció$_i$> helyén): <reláció$_i$> AS <sorváltozó>
A SELECT után elhelyezett attribútum-hivatkozásokhoz **oszlopváltozókat** rendelhetünk.
Szintaxis (SELECT után <reláció$_i$>.<attrib$_j$> helyén):
<reláció$_i$>.<attrib$_j$> AS <oszlopváltozó>
Így átnevezés lehetséges az eredmény megjelenítésekor:
Például:
SELECT név AS Filmszínház,
város AS Hely FROM mozi
$\Longrightarrow$

	Filmszínház	Hely
	⋮	

SQL Sor- és oszlopváltozók

A FROM után felsorolt relációkhoz **sorváltozókat** rendelhetünk.
Szintaxis (FROM után <reláció$_i$> helyén): <reláció$_i$> AS <sorváltozó>
A SELECT után elhelyezett attribútum-hivatkozásokhoz **oszlopváltozókat** rendelhetünk.
Szintaxis (SELECT után <reláció$_i$>.<attrib$_j$> helyén):
<reláció$_i$>.<attrib$_j$> AS <oszlopváltozó>
Így átnevezés lehetséges az eredmény megjelenítésekor:
Például:
SELECT név AS Filmszínház,
város AS Hely FROM mozi
$\Longrightarrow$

	Filmszínház	Hely
	⋮	

Az oszlopváltozók valójában csak az eredményreláció attribútumainak elnevezésére használhatók, a SELECT utasításon belül nem hivatkozhatunk rájuk.

SQL Sor- és oszlopváltozók

A FROM után felsorolt relációkhoz **sorváltozókat** rendelhetünk.
Szintaxis (FROM után <reláció$_i$> helyén): <reláció$_i$> AS <sorváltozó>
A SELECT után elhelyezett attribútum-hivatkozásokhoz **oszlopváltozókat** rendelhetünk.
Szintaxis (SELECT után <reláció$_i$>.<attrib$_j$> helyén):
<reláció$_i$>.<attrib$_j$> AS <oszlopváltozó>
Így átnevezés lehetséges az eredmény megjelenítésekor:
Például:
SELECT név AS Filmszínház,
város AS Hely FROM mozi
$\Longrightarrow$

	Filmszínház	Hely
	⋮	

Az oszlopváltozók valójában csak az eredményreláció attribútumainak elnevezésére használhatók, a SELECT utasításon belül nem hivatkozhatunk rájuk.
A <reláció$_i$>. előtag elhagyható, ha egyértelmű, hogy melyik relációról van szó, továbbá a <reláció$_i$>. előtag helyett <sorváltozó>. előtag is szerepeltethető.

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- <sorváltozó>.<attribútum> (mindig használható)

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- <sorváltozó>.<attribútum> (mindig használható)

Példa 11: A pénteken vetített filmek címei és rendezői (természetes illesztés)

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- <sorváltozó>.<attribútum> (mindig használható)

Példa 13: A pénteken vetített filmek címei és rendezői (természetes illesztés)
SELECT cím, rendező FROM film, vetít WHERE vetít.filmID = film.filmID AND nap='péntek'

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- <sorváltozó>.<attribútum> (mindig használható)

Példa 15: A pénteken vetített filmek címei és rendezői (természetes illesztés)
SELECT cím, rendező FROM film, vetít WHERE vetít.filmID = film.filmID AND nap='péntek'
Példa 16: Azok a várospárok, ahol vannak azonos nevű mozik

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- <sorváltozó>.<attribútum> (mindig használható)

Példa 17: A pénteken vetített filmek címei és rendezői (természetes illesztés)
SELECT cím, rendező FROM film, vetít WHERE vetít.filmID = film.filmID AND nap='péntek'
Példa 18: Azok a várospárok, ahol vannak azonos nevű mozik
SELECT m1.város, m2.város FROM mozi AS m1, mozi AS m2 WHERE m1.név = m2.név AND m1.város <> m2.város

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- <sorváltozó>.<attribútum> (mindig használható)

Példa 19: A pénteken vetített filmek címei és rendezői (természetes illesztés)
SELECT cím, rendező FROM film, vetít WHERE vetít.filmID = film.filmID AND nap='péntek'
Példa 20: Azok a várospárok, ahol vannak azonos nevű mozik
SELECT m1.város, m2.város FROM mozi AS m1, mozi AS m2 WHERE m1.név = m2.név AND m1.város <> m2.város
Megjegyzés: a várospárok mindkét sorrendben megjelennek, és több azonos nevű mozi esetén többször is megjelennek.

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- <attribútum> (ha ez egyértelmű)
- <reláció>.<attribútum> (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- <sorváltozó>.<attribútum> (mindig használható)

Példa 21: A pénteken vetített filmek címei és rendezői (természetes illesztés)
SELECT cím, rendező FROM film, vetít WHERE vetít.filmID = film.filmID AND nap='péntek'
Példa 22: Azok a várospárok, ahol vannak azonos nevű mozik
SELECT m1.város, m2.város FROM mozi AS m1, mozi AS m2 WHERE m1.név = m2.név AND m1.város <> m2.város
Megjegyzés: a várospárok mindkét sorrendben megjelennek, és több azonos nevű mozi esetén többször is megjelennek.
Az elsőre megoldás: <> helyett legyen <, amúgy meg DISTINCT
SELECT DISTINCT m1.város, m2.város FROM mozi AS m1, mozi AS m2 WHERE m1.név = m2.név AND m1.város < m2.város

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, ...

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, ...
- karakterlánc műveletek, összehasonlítás: CONCAT (||), LENGTH, LOWER, SUBSRT, SOUNDEX,...

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, ...
- karakterlánc műveletek, összehasonlítás: CONCAT (||), LENGTH, LOWER, SUBSRT, SOUNDEX,...
- halmazba tartozás: IN (halmaz), ...

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, ...
- karakterlánc műveletek, összehasonlítás: CONCAT (||), LENGTH, LOWER, SUBSRT, SOUNDEX,...
- halmazba tartozás: IN (halmaz), ...
- változóhivatkozások: <sorváltozó>.<attribútum>, <reláció>.<attribútum>, <attribútum>

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, . . .
- karakterlánc műveletek, összehasonlítás: CONCAT (||), LENGTH, LOWER, SUBSRT, SOUNDEX,. . .
- halmazba tartozás: IN (halmaz), . . .
- változóhivatkozások: <sorváltozó>.<attribútum>, <reláció>.<attribútum>, <attribútum>
- konstans (szám,karakterlánc): 137, 42e-3, 'füzér', . . .

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, . . .
- karakterlánc műveletek, összehasonlítás: CONCAT (||), LENGTH, LOWER, SUBSRT, SOUNDEX,. . .
- halmazba tartozás: IN (halmaz), . . .
- változóhivatkozások: <sorváltozó>.<attribútum>, <reláció>.<attribútum>, <attribútum>
- konstans (szám,karakterlánc): 137, 42e-3, 'füzér', . . .
- NULL érték vizsgálata: IS NULL, IS NOT NULL (később lesz)

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, . . .
- karakterlánc műveletek, összehasonlítás: CONCAT (||), LENGTH, LOWER, SUBSRT, SOUNDEX,. . .
- halmazba tartozás: IN (halmaz), . . .
- változóhivatkozások: <sorváltozó>.<attribútum>, <reláció>.<attribútum>, <attribútum>
- konstans (szám,karakterlánc): 137, 42e-3, 'füzér', . . .
- NULL érték vizsgálata: IS NULL, IS NOT NULL (később lesz)
- alkérdés is lehet itt: (majd erről később)

LIKE és BETWEEN használata

LIKE használata:

- '_' egy tetszőleges karakterre illeszkedik

LIKE és BETWEEN használata

LIKE használata:

- '_' egy tetszőleges karakterre illeszkedik
- '%' tetszőleges karakterláncra illeszkedik

LIKE és BETWEEN használata

LIKE használata:

- '_' egy tetszőleges karakterre illeszkedik
- '%' tetszőleges karakterláncra illeszkedik

BETWEEN használata: BETWEEN a AND b jelentése $a \leq . \leq b$

LIKE és BETWEEN használata

LIKE használata:

- '_' egy tetszőleges karakterre illeszkedik
- '%' tetszőleges karakterláncra illeszkedik

BETWEEN használata: BETWEEN a AND b jelentése $a \leq . \leq b$
Példa 26: A 150 és 200 közötti azonosítójú mozik közül azok, amelyek B-vel kezdődő nevű városban vannak, és a nevük hárombetűs.

LIKE és BETWEEN használata

LIKE használata:

- '_' egy tetszőleges karakterre illeszkedik
- '%' tetszőleges karakterláncra illeszkedik

BETWEEN használata: BETWEEN a AND b jelentése $a \leq . \leq b$
Példa 27: A 150 és 200 közötti azonosítójú mozik közül azok, amelyek B-vel kezdődő nevű városban vannak, és a nevük hárombetűs.

SELECT név FROM mozi WHERE moziID BETWEEN 150 AND 200 AND város LIKE 'B%' AND név LIKE '_ _ _'

Műveletek relációkkal

A részeredményül kapott relációkkal (**ha azok sémája lényegében azonos!**) halmazműveleteket (unió, metszet, különbség) végezhetünk.

Műveletek relációkkal

A részeredményül kapott relációkkal (**ha azok sémája lényegében azonos!**) halmazműveleteket (unió, metszet, különbség) végezhetünk.

Unió (valamely eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> UNION <eredményreláció2>

Műveletek relációkkal

A részeredményül kapott relációkkal (**ha azok sémája lényegében azonos!**) halmazműveleteket (unió, metszet, különbség) végezhetünk.

Unió (valamely eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> UNION <eredményreláció2>
- Példa 32: A pénteken vagy szombaton játszott filmek :

Műveletek relációkkal

A részeredményül kapott relációkkal (**ha azok sémája lényegében azonos!**) halmazműveleteket (unió, metszet, különbség) végezhetünk.

Unió (valamely eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> UNION <eredményreláció2>
- Példa 34: A pénteken vagy szombaton játszott filmek :
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  UNION
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

  (nem hatékony!)

Műveletek relációkkal

A részeredményül kapott relációkkal (**ha azok sémája lényegében azonos!**) halmazműveleteket (unió, metszet, különbség) végezhetünk.

Unió (valamely eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> UNION <eredményreláció2>
- Példa 36: A pénteken vagy szombaton játszott filmek :
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  UNION
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

  (nem hatékony!)

Metszet (mindkét eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> INTERSECT <eredményreláció2>

Műveletek relációkkal

A részeredményül kapott relációkkal (**ha azok sémája lényegében azonos!**) halmazműveleteket (unió, metszet, különbség) végezhetünk.

Unió (valamely eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> UNION <eredményreláció2>
- Példa 38: A pénteken vagy szombaton játszott filmek :
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  UNION
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

  (nem hatékony!)

Metszet (mindkét eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> INTERSECT <eredményreláció2>
- Példa 39: A pénteken és szombaton is játszott filmek:

Műveletek relációkkal

A részeredményül kapott relációkkal (**ha azok sémája lényegében azonos!**) halmazműveleteket (unió, metszet, különbség) végezhetünk.

Unió (valamely eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> UNION <eredményreláció2>
- Példa 40: A pénteken vagy szombaton játszott filmek :
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  UNION
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

  (nem hatékony!)

Metszet (mindkét eredményrelációban szereplő sorok):

- Szintaxis: <eredményreláció1> INTERSECT <eredményreláció2>
- Példa 41: A pénteken és szombaton is játszott filmek:
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  INTERSECT
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

Műveletek relációkkal

Különbség (az első reláció azon sorai, melyek a másodikban nem szerepelnek):

- Szintaxis: <eredményreláció1> MINUS <eredményreláció2>

Műveletek relációkkal

Különbség (az első reláció azon sorai, melyek a másodikban nem szerepelnek):

- Szintaxis: <eredményreláció1> MINUS <eredményreláció2>
- Példa 43: A pénteken igen, de szombaton nem játszott filmek:

Műveletek relációkkal

Különbség (az első reláció azon sorai, melyek a másodikban nem szerepelnek):

- Szintaxis: <eredményreláció1> MINUS <eredményreláció2>
- Példa 44: A pénteken igen, de szombaton nem játszott filmek:
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  MINUS
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

Műveletek relációkkal

Különbség (az első reláció azon sorai, melyek a másodikban nem szerepelnek):

- Szintaxis: <eredményreláció1> MINUS <eredményreláció2>
- Példa 45: A pénteken igen, de szombaton nem játszott filmek:
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  MINUS
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

A szabványban MINUS helyett EXCEPT szerepel, de a gyakorlatban a MINUS használatos.

Műveletek relációkkal

Különbség (az első reláció azon sorai, melyek a másodikban nem szerepelnek):

- Szintaxis: <eredményreláció1> MINUS <eredményreláció2>
- Példa 46: A pénteken igen, de szombaton nem játszott filmek:
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
  MINUS
  (SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

A szabványban MINUS helyett EXCEPT szerepel, de a gyakorlatban a MINUS használatos.

Állítás

Az SQL relációsan teljes.

Állítás

Az SQL relációsan teljes.

Bizonyítás.

Most láttuk az uniót és különbséget, a többi pedig már volt, de újra:

Állítás

Az SQL relációsan teljes.

Bizonyítás.

Most láttuk az uniót és különbséget, a többi pedig már volt, de újra:

vetítés: $\pi_{A_{i_1},A_{i_2},\ldots,A_{i_k}}(R)$-nek megfelelő lekérdezés: SELECT $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ FROM R

Állítás

Az SQL relációsan teljes.

Bizonyítás.

Most láttuk az uniót és különbséget, a többi pedig már volt, de újra:

vetítés: $\pi_{A_{i_1},A_{i_2},\ldots,A_{i_k}}(R)$-nek megfelelő lekérdezés: SELECT $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ FROM R

kiválasztás: $\sigma_F(R)$-nek megfelel a
SELECT * FROM R WHERE F'
ahol F'az, ami F-ből jön átírással ($\wedge, \vee, \neg$ helyett AND, OR, NOT)

Állítás

Az SQL relációsan teljes.

Bizonyítás.

Most láttuk az uniót és különbséget, a többi pedig már volt, de újra:

vetítés: $\pi_{A_{i_1},A_{i_2},\ldots,A_{i_k}}(R)$-nek megfelelő lekérdezés: SELECT $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ FROM R

kiválasztás: $\sigma_F(R)$-nek megfelel a
SELECT * FROM R WHERE F'
ahol F'az, ami F-ből jön átírással ($\wedge, \vee, \neg$ helyett AND, OR, NOT)

szorzat: SELECT $R.A_1, R.A_2, \ldots, R.A_k, S.B_1, \ldots, S.B_l$ FROM R,S $\surd$ $\square$

Multihalmazok-halmazok

- Az SQL alapértelmezésben nem tünteti el a többszörös sorokat, kivétel: UNION, INTERSECT, EXCEPT, ennél a háromnál eltűnnek az ismétlődések

Multihalmazok-halmazok

- Az SQL alapértelmezésben nem tünteti el a többszörös sorokat, kivétel: UNION, INTERSECT, EXCEPT, ennél a háromnál eltűnnek az ismétlődések
- Ha el akarjuk tüntetni az ismétlődéseket:
  SELECT DISTINCT

Multihalmazok-halmazok

- Az SQL alapértelmezésben nem tünteti el a többszörös sorokat, kivétel: UNION, INTERSECT, EXCEPT, ennél a háromnál eltűnnek az ismétlődések
- Ha el akarjuk tüntetni az ismétlődéseket:
  SELECT DISTINCT
- Ha a halmazműveleteknél mégsem akarom eltüntetni az ismétlődéseket:
  UNION ALL, EXCEPT ALL, INTERSECT ALL

Multihalmazok-halmazok

- Az SQL alapértelmezésben nem tünteti el a többszörös sorokat, kivétel: UNION, INTERSECT, EXCEPT, ennél a háromnál eltűnnek az ismétlődések
- Ha el akarjuk tüntetni az ismétlődéseket:
  SELECT DISTINCT
- Ha a halmazműveleteknél mégsem akarom eltüntetni az ismétlődéseket:
  UNION ALL, EXCEPT ALL, INTERSECT ALL
- Nem (mindig) éri meg közben is törekedni arra, hogy ne legyen ismétlődés, elég a végén, mert:

Multihalmazok-halmazok

- Az SQL alapértelmezésben nem tünteti el a többszörös sorokat, kivétel: UNION, INTERSECT, EXCEPT, ennél a háromnál eltűnnek az ismétlődések
- Ha el akarjuk tüntetni az ismétlődéseket:
  SELECT DISTINCT
- Ha a halmazműveleteknél mégsem akarom eltüntetni az ismétlődéseket:
  UNION ALL, EXCEPT ALL, INTERSECT ALL
- Nem (mindig) éri meg közben is törekedni arra, hogy ne legyen ismétlődés, elég a végén, mert:
  Az ismétlődés kiküszöbölése sok munka, mert rendezni kell az egész relációt hozzá.

Aggregátumok

- Aggregátumok számolása: SUM, MIN, MAX, AVG, COUNT,. . .

Aggregátumok

- Aggregátumok számolása: SUM, MIN, MAX, AVG, COUNT,. . .
- Az, hogy COUNT hogyan kezeli a többszörös sorokat, az rendszerfüggő.
  Ha biztosra akarunk menni: COUNT (DISTINCT <attribútum>), COUNT (ALL <attribútum>)

Aggregátumok

- Aggregátumok számolása: SUM, MIN, MAX, AVG, COUNT,...
- Az, hogy COUNT hogyan kezeli a többszörös sorokat, az rendszerfüggő. Ha biztosra akarunk menni: COUNT (DISTINCT <attribútum>), COUNT (ALL <attribútum>)
- Lehetőségünk van bizonyos attribútumok értéke szerint csoportosítani az eredményt, és így aggregált sorokat képezni.

Aggregátumok

- Aggregátumok számolása: SUM, MIN, MAX, AVG, COUNT,. . .
- Az, hogy COUNT hogyan kezeli a többszörös sorokat, az rendszerfüggő. Ha biztosra akarunk menni: COUNT (DISTINCT <attribútum>), COUNT (ALL <attribútum>)
- Lehetőségünk van bizonyos attribútumok értéke szerint csoportosítani az eredményt, és így aggregált sorokat képezni.

Erre az utóbbira példa a következő reláció:

MOZI

moziID	név	város	székszám
1	Corvin	Budapest	2500
2	Elit	Sopron	300
3	Sopron Plaza Megaflex	Sopron	2000
4	Szindbád	Budapest	600
5	Tabán	Budapest	200
6	Uránia	Pécs	500

Aggregátumok

Csoportosítsunk a város attribútum szerint:

MOZI

moziID	név	város	székszám
1	Corvin	Budapest	2500
4	Szindbád	Budapest	600
5	Tabán	Budapest	200
6	Uránia	Pécs	500
2	Elit	Sopron	300
3	Sopron Plaza Megaflex	Sopron	2000

Aggregátumok

Csoportosítsunk a város attribútum szerint:

MOZI	moziID	név	város	székszám
	1	Corvin	Budapest	2500
	4	Szindbád	Budapest	600
	5	Tabán	Budapest	200
	6	Uránia	Pécs	500
	2	Elit	Sopron	300
	3	Sopron Plaza Megaflex	Sopron	2000

Képezzük minden városra a székszámok összegét:

MOZI	város	össz_székszám
	Budapest	3300
	Pécs	500
	Sopron	2300

Példa 48: Mindez SQL-ben

Aggregátumok

Csoportosítsunk a város attribútum szerint:

MOZI	moziID	név	város	székszám
	1	Corvin	Budapest	2500
	4	Szindbád	Budapest	600
	5	Tabán	Budapest	200
	6	Uránia	Pécs	500
	2	Elit	Sopron	300
	3	Sopron Plaza Megaflex	Sopron	2000

Képezzük minden városra a székszámok összegét:

MOZI	város	össz_székszám
	Budapest	3300
	Pécs	500
	Sopron	2300

Példa 49: Mindez SQL-ben

SELECT város, SUM(székszám) AS össz_székszám FROM mozi GROUP BY város

Aggregátumok

Példa 50: Az egyes városok legkisebb és legnagyobb mozijának mérete

Aggregátumok

Példa 53: Az egyes városok legkisebb és legnagyobb mozijának mérete
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város

Aggregátumok

Példa 56: Az egyes városok legkisebb és legnagyobb mozijának mérete
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város
Példák, ahol nincs csoportosítás:
Példa 57: A létező legnagyobb és a legkisebb székszám

Aggregátumok

Példa 59: Az egyes városok legkisebb és legnagyobb mozijának mérete
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város
Példák, ahol nincs csoportosítás:
Példa 60: A létező legnagyobb és a legkisebb székszám
SELECT MIN(székszám), MAX(székszám) FROM mozi

Aggregátumok

Példa 62: Az egyes városok legkisebb és legnagyobb mozijának mérete
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város
Példák, ahol nincs csoportosítás:
Példa 63: A létező legnagyobb és a legkisebb székszám
SELECT MIN(székszám), MAX(székszám) FROM mozi
Példa 64: Az összes székszám

Aggregátumok

Példa 65: Az egyes városok legkisebb és legnagyobb mozijának mérete
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város
Példák, ahol nincs csoportosítás:
Példa 66: A létező legnagyobb és a legkisebb székszám
SELECT MIN(székszám), MAX(székszám) FROM mozi
Példa 67: Az összes székszám
SELECT SUM(székszám) FROM mozi

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá),

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk,

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk, majd kiszámoljuk minden csoportra az aggregátumot és kiírjuk.

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk, majd kiszámoljuk minden csoportra az aggregátumot és kiírjuk.
- Amennyiben aggregátumokat képzünk a GROUP BY segítségével, akkor csak azokra az attribútumokra hivatkozhatunk közvetlenül a SELECT-ben, ami szerint csoportosítottunk.

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk, majd kiszámoljuk minden csoportra az aggregátumot és kiírjuk.
- Amennyiben aggregátumokat képzünk a GROUP BY segítségével, akkor csak azokra az attribútumokra hivatkozhatunk közvetlenül a SELECT-ben, ami szerint csoportosítottunk. Ezen attribútumok értékei ugyanis egy aggregátumon belül jól meghatározottak. A többi attribútum az aggregátumon belül többféle értéket is felvehet. Ezért rájuk csak oszlopfüggvényeken (aggregátumokon) keresztül hivatkozhatunk.

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk, majd kiszámoljuk minden csoportra az aggregátumot és kiírjuk.
- Amennyiben aggregátumokat képzünk a GROUP BY segítségével, akkor csak azokra az attribútumokra hivatkozhatunk közvetlenül a SELECT-ben, ami szerint csoportosítottunk. Ezen attribútumok értékei ugyanis egy aggregátumon belül jól meghatározottak. A többi attribútum az aggregátumon belül többféle értéket is felvehet. Ezért rájuk csak oszlopfüggvényeken (aggregátumokon) keresztül hivatkozhatunk.
- Lehet több oszlop szerint is GROUP BY, ekkor azok a sorok lesznek egy csoportban, ahol mindegyik GROUP BY után felsorolt oszlop értéke megegyezik.
- Lehet GROUP BY aggregátum nélkül is

Példa 73:
SELECT város FROM mozi GROUP BY város

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk, majd kiszámoljuk minden csoportra az aggregátumot és kiírjuk.
- Amennyiben aggregátumokat képzünk a GROUP BY segítségével, akkor csak azokra az attribútumokra hivatkozhatunk közvetlenül a SELECT-ben, ami szerint csoportosítottunk. Ezen attribútumok értékei ugyanis egy aggregátumon belül jól meghatározottak. A többi attribútum az aggregátumon belül többféle értéket is felvehet. Ezért rájuk csak oszlopfüggvényeken (aggregátumokon) keresztül hivatkozhatunk.
- Lehet több oszlop szerint is GROUP BY, ekkor azok a sorok lesznek egy csoportban, ahol mindegyik GROUP BY után felsorolt oszlop értéke megegyezik.
- Lehet GROUP BY aggregátum nélkül is

  Példa 74:
  SELECT város FROM mozi GROUP BY város

  Kiírja az összes várost (pontosan egyszer), ahol van mozi.

Aggregátumok

- Kiértékelés: Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk, majd kiszámoljuk minden csoportra az aggregátumot és kiírjuk.
- Amennyiben aggregátumokat képzünk a GROUP BY segítségével, akkor csak azokra az attribútumokra hivatkozhatunk közvetlenül a SELECT-ben, ami szerint csoportosítottunk. Ezen attribútumok értékei ugyanis egy aggregátumon belül jól meghatározottak. A többi attribútum az aggregátumon belül többféle értéket is felvehet. Ezért rájuk csak oszlopfüggvényeken (aggregátumokon) keresztül hivatkozhatunk.
- Lehet több oszlop szerint is GROUP BY, ekkor azok a sorok lesznek egy csoportban, ahol mindegyik GROUP BY után felsorolt oszlop értéke megegyezik.
- Lehet GROUP BY aggregátum nélkül is

  Példa 75:
  SELECT város FROM mozi GROUP BY város

  Kiírja az összes várost (pontosan egyszer), ahol van mozi.
  Ugyanaz, mint a
  SELECT DISTINCT város FROM mozi

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.
Példa 77: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.
Példa 78: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város HAVING COUNT(név)>1

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.
Példa 79: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város HAVING COUNT(név)>1

- a csoportra vonatkozó feltételt a HAVING kulcsszó vezeti be

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.
Példa 80: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város HAVING COUNT(név)>1

- a csoportra vonatkozó feltételt a HAVING kulcsszó vezeti be
- olyan feltételt írunk ide, ami csoportra vonatkozik (különben WHERE-be írnánk)

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.
Példa 81: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város HAVING COUNT(név)>1

- a csoportra vonatkozó feltételt a HAVING kulcsszó vezeti be
- olyan feltételt írunk ide, ami csoportra vonatkozik (különben WHERE-be írnánk)
- csak GROUP BY-jal együtt használható

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.
Példa 82: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város HAVING COUNT(név)>1

- a csoportra vonatkozó feltételt a HAVING kulcsszó vezeti be
- olyan feltételt írunk ide, ami csoportra vonatkozik (különben WHERE-be írnánk)
- csak GROUP BY-jal együtt használható
- a kiértékelés során a csoportosítás után minden egyes csoportra megnézzük a feltételt és eldobjuk azokat a csoportokat, amikre a feltétel nem áll és a maradékkal dolgozunk tovább

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.
Példa 83: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város HAVING COUNT(név)>1

- a csoportra vonatkozó feltételt a HAVING kulcsszó vezeti be
- olyan feltételt írunk ide, ami csoportra vonatkozik (különben WHERE-be írnánk)
- csak GROUP BY-jal együtt használható
- a kiértékelés során a csoportosítás után minden egyes csoportra megnézzük a feltételt és eldobjuk azokat a csoportokat, amikre a feltétel nem áll és a maradékkal dolgozunk tovább
- HAVING megkerülhető, mindent, amit lehet HAVING-gel, lehet máshogy is (majd lesz erről szó az alkérdéseknél)

Adatbázisok elmélete

SQL folytatás

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

10. előadás

A hat alapkulcsszó

- SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY

A hat alapkulcsszó

- SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
- Ebben a sorrendben jönnek

A hat alapkulcsszó

- SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
- Ebben a sorrendben jönnek
- SELECT és FROM kell, a többi opcionális

A hat alapkulcsszó

- SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
- Ebben a sorrendben jönnek
- SELECT és FROM kell, a többi opcionális
- HAVING csak GROUP BY-jal

Alkérdések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.

Alkérdések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.
- Tipikusan WHERE feltételében áll, ezáltal sokkal összetettebb kiválasztási feltételek jönnek létre, mint a relációs algebrában.

Alkérdések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.
- Tipikusan WHERE feltételében áll, ezáltal sokkal összetettebb kiválasztási feltételek jönnek létre, mint a relációs algebrában.

Alkérdés FROM záradékban

Alkérdések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.
- Tipikusan WHERE feltételében áll, ezáltal sokkal összetettebb kiválasztási feltételek jönnek létre, mint a relációs algebrában.

Alkérdés FROM záradékban

A kiválasztáshoz használt relációk lehetnek alkérdés által származtatott relációk is.

Alkérdések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.
- Tipikusan WHERE feltételében áll, ezáltal sokkal összetettebb kiválasztási feltételek jönnek létre, mint a relációs algebrában.

Alkérdés FROM záradékban

A kiválasztáshoz használt relációk lehetnek alkérdés által származtatott relációk is.
Példa 88: A filmek címe, rendezője és a rendező filmjeinek száma

Alkérdések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.
- Tipikusan WHERE feltételében áll, ezáltal sokkal összetettebb kiválasztási feltételek jönnek létre, mint a relációs algebrában.

Alkérdés FROM záradékban

A kiválasztáshoz használt relációk lehetnek alkérdés által származtatott relációk is.
Példa 89: A filmek címe, rendezője és a rendező filmjeinek száma

```
SELECT f1.cím, f1.rendező, f2.filmszám FROM
film AS f1,
(SELECT rendező, COUNT(*) AS filmszám FROM film GROUP BY rendező) AS f2
WHERE f1.rendező = f2.rendező
```

Alkérdések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.
- Tipikusan WHERE feltételében áll, ezáltal sokkal összetettebb kiválasztási feltételek jönnek létre, mint a relációs algebrában.

Alkérdés FROM záradékban

A kiválasztáshoz használt relációk lehetnek alkérdés által származtatott relációk is.
Példa 90: A filmek címe, rendezője és a rendező filmjeinek száma

```
SELECT f1.cím, f1.rendező, f2.filmszám FROM
film AS f1,
(SELECT rendező, COUNT(*) AS filmszám FROM film GROUP BY rendező) AS f2
WHERE f1.rendező = f2.rendező
```

Vigyázat! Itt nem jött létre f2 nevű reláció, csak annyi történik, hogy az f2 nevű sorváltozó befutja az alkérdés eredményéül kapott reláció sorait. Egyszer kiszámolódik az alkérdés és ennek eredményét használjuk a továbbiakban.

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.
Ezeknek az attribútumok számában meg kell egyezniük az alkérdés eredményének oszlopszámával.

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.
Ezeknek az attribútumok számában meg kell egyezniük az alkérdés eredményének oszlopszámával.

- **Egyenlőség vizsgálata**
  Csak akkor lehetséges, ha az alkérdés egysoros relációt ír le (azaz az eredménye egyetlen érték vagy érték-vektor).

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.
Ezeknek az attribútumok számában meg kell egyezniük az alkérdés eredményének oszlopszámával.

- **Egyenlőség vizsgálata**
  Csak akkor lehetséges, ha az alkérdés egysoros relációt ír le (azaz az eredménye egyetlen érték vagy érték-vektor).
  Az egyenlőség fennáll, ha az adott attribútumok értékei megegyeznek az alkérdés által adott reláció megfelelő attribútumainak értékével.

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.
Ezeknek az attribútumok számában meg kell egyezniük az alkérdés eredményének oszlopszámával.

- **Egyenlőség vizsgálata**
  Csak akkor lehetséges, ha az alkérdés egysoros relációt ír le (azaz az eredménye egyetlen érték vagy érték-vektor).
  Az egyenlőség fennáll, ha az adott attribútumok értékei megegyeznek az alkérdés által adott reláció megfelelő attribútumainak értékével.
  Szintaxis:
  SELECT ... FROM ...
  WHERE (<attrib$_{11}$>, ..., <attrib$_{1n}$>) = (SELECT <attrib$_{21}$>, ..., <attrib$_{2n}$> FROM ...)
  A nem egyenlőség vizsgálatára a $<>$ használandó.

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.
Ezeknek az attribútumok számában meg kell egyezniük az alkérdés eredményének oszlopszámával.

- **Egyenlőség vizsgálata**
  Csak akkor lehetséges, ha az alkérdés egysoros relációt ír le (azaz az eredménye egyetlen érték vagy érték-vektor).
  Az egyenlőség fennáll, ha az adott attribútumok értékei megegyeznek az alkérdés által adott reláció megfelelő attribútumainak értékével.
  Szintaxis:
  SELECT ... FROM ...
  WHERE (<attrib$_{11}$>, ..., <attrib$_{1n}$>) = (SELECT <attrib$_{21}$>, ..., <attrib$_{2n}$> FROM ...)
  A nem egyenlőség vizsgálatára a $<>$ használandó.
  Példa 96: A legnagyobb mozik nevei

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.
Ezeknek az attribútumok számában meg kell egyezniük az alkérdés eredményének oszlopszámával.

- **Egyenlőség vizsgálata**
  Csak akkor lehetséges, ha az alkérdés egysoros relációt ír le (azaz az eredménye egyetlen érték vagy érték-vektor).
  Az egyenlőség fennáll, ha az adott attribútumok értékei megegyeznek az alkérdés által adott reláció megfelelő attribútumainak értékével.
  Szintaxis:
  SELECT ... FROM ...
  WHERE (<attrib$_{11}$>, ..., <attrib$_{1n}$>) = (SELECT <attrib$_{21}$>, ..., <attrib$_{2n}$> FROM ...)
  A nem egyenlőség vizsgálatára a $<>$ használandó.
  Példa 97: A legnagyobb mozik nevei
  SELECT név FROM mozi
  WHERE mozi.székszám = (SELECT MAX(székszám) FROM mozi)

Alkérdés WHERE záradékban

- **Tartalmazás vizsgálata**
  Több sort adó alkérdésre is értelmezett.

Alkérdés WHERE záradékban

- **Tartalmazás vizsgálata**
  Több sort adó alkérdésre is értelmezett.
  A tartalmazás fennáll, ha a vizsgált attribútumok értéke megegyezik az alkérdés eredményének valamely sorával.

Alkérdés WHERE záradékban

- **Tartalmazás vizsgálata**
  Több sort adó alkérdésre is értelmezett.
  A tartalmazás fennáll, ha a vizsgált attribútumok értéke megegyezik az alkérdés eredményének valamely sorával.
  Szintaxis:
  SELECT ... FROM ...
  WHERE (<attrib$_{11}$>, ..., <attrib$_{1n}$>) IN (SELECT <attrib$_{21}$>, ..., <attrib$_{2n}$> FROM ...)

Alkérdés WHERE záradékban

- **Tartalmazás vizsgálata**
  Több sort adó alkérdésre is értelmezett.
  A tartalmazás fennáll, ha a vizsgált attribútumok értéke megegyezik az alkérdés eredményének valamely sorával.
  Szintaxis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) IN (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)
  A nem tartalmazás vizsgálatára a **NOT IN** használandó.

  Példa 101: A nem vetített filmek címe és rendezője

Alkérdés WHERE záradékban

- **Tartalmazás vizsgálata**
  Több sort adó alkérdésre is értelmezett.
  A tartalmazás fennáll, ha a vizsgált attribútumok értéke megegyezik az alkérdés eredményének valamely sorával.
  Szintaxis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) IN (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)
  A nem tartalmazás vizsgálatára a **NOT IN** használandó.

  Példa 102: A nem vetített filmek címe és rendezője
  SELECT cím, rendező FROM film AS f1
  WHERE f1.filmID NOT IN (SELECT v1.filmID FROM vetít AS v1)

Alkérdések

- **Alkérdés valamely vagy minden sorának vizsgálata**
  Tipikusan több sort szolgáltató alkérdés esetén, valamilyen összehasonlító operátorral együtt használatosak az **ANY** és az **ALL** kulcsszavak.

Alkérdések

- **Alkérdés valamely vagy minden sorának vizsgálata**
  Tipikusan több sort szolgáltató alkérdés esetén, valamilyen összehasonlító operátorral együtt használatosak az **ANY** és az **ALL** kulcsszavak.
  Az **ANY**-t (**ALL**-t) tartalmazó feltétel teljesül, ha a vizsgált attribútumok értéke és az alkérdés valamely (minden) sorára az összehasonlító operátor igaz értéket ad.

Alkérdések

- **Alkérdés valamely vagy minden sorának vizsgálata**
  Tipikusan több sort szolgáltató alkérdés esetén, valamilyen összehasonlító operátorral együtt használatosak az **ANY** és az **ALL** kulcsszavak.
  Az **ANY**-t (**ALL**-t) tartalmazó feltétel teljesül, ha a vizsgált attribútumok értéke és az alkérdés valamely (minden) sorára az összehasonlító operátor igaz értéket ad.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) <op> [ANY | ALL] (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)

Alkérdések

- **Alkérdés valamely vagy minden sorának vizsgálata**
  Tipikusan több sort szolgáltató alkérdés esetén, valamilyen összehasonlító operátorral együtt használatosak az **ANY** és az **ALL** kulcsszavak.
  Az **ANY**-t (**ALL**-t) tartalmazó feltétel teljesül, ha a vizsgált attribútumok értéke és az alkérdés valamely (minden) sorára az összehasonlító operátor igaz értéket ad.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) <op> [ANY | ALL] (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)
  A „semelyik”, illetve a „nem mind” leírására a **NOT ANY**, illetve a **NOT ALL** használatosak.

Alkérdések

- **Alkérdés valamely vagy minden sorának vizsgálata**
  Tipikusan több sort szolgáltató alkérdés esetén, valamilyen összehasonlító operátorral együtt használatosak az **ANY** és az **ALL** kulcsszavak.
  Az **ANY**-t (**ALL**-t) tartalmazó feltétel teljesül, ha a vizsgált attribútumok értéke és az alkérdés valamely (minden) sorára az összehasonlító operátor igaz értéket ad.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) <op> [ANY | ALL] (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)
  A „semelyik”, illetve a „nem mind” leírására a **NOT ANY**, illetve a **NOT ALL** használatosak.

Példa 107: Ismét a legnagyobb mozi(k)

Alkérdések

- **Alkérdés valamely vagy minden sorának vizsgálata**
  Tipikusan több sort szolgáltató alkérdés esetén, valamilyen összehasonlító operátorral együtt használatosak az **ANY** és az **ALL** kulcsszavak.
  Az **ANY**-t (**ALL**-t) tartalmazó feltétel teljesül, ha a vizsgált attribútumok értéke és az alkérdés valamely (minden) sorára az összehasonlító operátor igaz értéket ad.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) <op> [ANY | ALL] (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)
  A „semelyik”, illetve a „nem mind” leírására a **NOT ANY**, illetve a **NOT ALL** használatosak.

Példa 108: Ismét a legnagyobb mozi(k)

```
SELECT m2.város, m2.név, m2.székszám FROM mozi AS m2
WHERE m2.székszám >= ALL (SELECT m1.székszám FROM mozi AS m1)
```

- **Alkérdés ürességének vizsgálata**
  Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérdés által leírt relációnak.

- **Alkérdés ürességének vizsgálata**
  Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérdés által leírt relációnak.
  Az ezt tartalmazó feltétel teljesül, ha van legalább egy sor.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE EXISTS (SELECT <attrib$_1$>, . . . , <attrib$_n$> FROM . . .)

- **Alkérdés ürességének vizsgálata**
  Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérdés által leírt relációnak.
  Az ezt tartalmazó feltétel teljesül, ha van legalább egy sor.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE EXISTS (SELECT <attrib$_1$>, . . . , <attrib$_n$> FROM . . .)
  A nem létezés vizsgálatára a **NOT EXISTS** használandó.

- **Alkérdés ürességének vizsgálata**
  Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérdés által leírt relációnak.
  Az ezt tartalmazó feltétel teljesül, ha van legalább egy sor.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE EXISTS (SELECT <attrib$_1$>, . . . , <attrib$_n$> FROM . . .)
  A nem létezés vizsgálatára a **NOT EXISTS** használandó.

  Példa 112: Azok a városok, ahol van legalább két mozi

- **Alkérdés ürességének vizsgálata**
  Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérdés által leírt relációnak.
  Az ezt tartalmazó feltétel teljesül, ha van legalább egy sor.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE EXISTS (SELECT <attrib$_1$>, . . . , <attrib$_n$> FROM . . .)
  A nem létezés vizsgálatára a **NOT EXISTS** használandó.

  Példa 113: Azok a városok, ahol van legalább két mozi
  SELECT m1.város FROM mozi AS m1
  WHERE EXISTS (SELECT * FROM mozi AS m2 WHERE m1.város =m2.város
  AND m1.név<>m2.név)

- **Alkérdés ürességének vizsgálata**
  Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérdés által leírt relációnak.
  Az ezt tartalmazó feltétel teljesül, ha van legalább egy sor.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE EXISTS (SELECT <attrib$_1$>, . . . , <attrib$_n$> FROM . . .)
  A nem létezés vizsgálatára a **NOT EXISTS** használandó.

  Példa 114: Azok a városok, ahol van legalább két mozi
  SELECT m1.város FROM mozi AS m1
  WHERE EXISTS (SELECT * FROM mozi AS m2 WHERE m1.város =m2.város
  AND m1.név<>m2.név)

  Ez úgy nevezett korrelált alkérdés:
  ennek kiértékelése során minden egyes lehetséges értékére az m1 sorváltozónak lefut az alkérdés és kiírás van, ha az alkérdés eredménye nem üres.
  A korábbi esetekben csak egyszer kellett kiértékelni az alkérdést, itt annyiszor, ahány sora a mozi-nak van.

HAVING megkerülése alkérdéssel

Nézzük egy példán, de általában is így megy:
HAVING-gel:

Azokra a városokra számolunk legkisebb és legnagyobb mozit, ahol van legalább 2 mozi

```
SELECT város, MIN(székszám), MAX(székszám)
FROM mozi
GROUP BY város
HAVING COUNT(*)>1
```

HAVING megkerülése alkérdéssel

Nézzük egy példán, de általában is így megy:
HAVING-gel:

Azokra a városokra számolunk legkisebb és legnagyobb mozit, ahol van legalább 2 mozi

```
SELECT város, MIN(székszám), MAX(székszám)
FROM mozi
GROUP BY város
HAVING COUNT(*)>1
```

HAVING nélkül, alkérdéssel:

HAVING megkerülése alkérdéssel

Nézzük egy példán, de általában is így megy:
HAVING-gel:

Azokra a városokra számolunk legkisebb és legnagyobb mozit, ahol van legalább 2 mozi

```
SELECT város, MIN(székszám), MAX(székszám)
FROM mozi
GROUP BY város
HAVING COUNT(*)>1
```

HAVING nélkül, alkérdéssel:

```
SELECT város, minszékszám, maxszékszám
FROM (SELECT város, MIN(székszám) AS minszékszám, MAX(székszám)
                              AS maxszékszám, COUNT(*) AS darab
                    FROM mozi
                     GROUP BY város)
WHERE darab >1
```

NULL érték az SQL-ben

Az SQL-ben az ismeretlen vagy nem létező értéket a **NULL** érték jelképezi.
A **NULL** használatakor ügyelni kell rá, hogy az aritmetikai és összehasonlító operátorok speciálisan értelmezettek rá.

NULL érték az SQL-ben

Az SQL-ben az ismeretlen vagy nem létező értéket a **NULL** érték jelképezi.
A **NULL** használatakor ügyelni kell rá, hogy az aritmetikai és összehasonlító operátorok speciálisan értelmezettek rá.
Például:

NULL * 0 értéke nem 0, hanem **NULL**.

NULL érték az SQL-ben

Az SQL-ben az ismeretlen vagy nem létező értéket a **NULL** érték jelképezi.
A **NULL** használatakor ügyelni kell rá, hogy az aritmetikai és összehasonlító operátorok speciálisan értelmezettek rá.
Például:

NULL * 0 értéke nem 0, hanem **NULL**.

rendező = NULL értéke nem IGAZ, nem HAMIS, hanem a logikai ISMERETLEN érték

$$\Longrightarrow \text{UN.}$$

Háromértékű logika

	$\neg$
I	H
H	I
UN	UN

Háromértékű logika

	$\neg$
I	H
H	I
UN	UN

$\vee$	I	H	UN
I	I	I	I
H	I	H	UN
UN	I	UN	UN

Háromértékű logika

	$\neg$
I	H
H	I
UN	UN

$\vee$	I	H	UN
I	I	I	I
H	I	H	UN
UN	I	UN	UN

$\wedge$	I	H	UN
I	I	H	UN
H	H	H	H
UN	UN	H	UN

Háromértékű logika

	$\neg$
I	H
H	I
UN	UN

$\vee$	I	H	UN
I	I	I	I
H	I	H	UN
UN	I	UN	UN

$\wedge$	I	H	UN
I	I	H	UN
H	H	H	H
UN	UN	H	UN

Tehát egy állítás nem csak igaz vagy hamis lehet, hanem „ismeretlen” is és egy WHERE-beli állításnál természetesen csak az számit találatnak, ha igaz az állítás, az „ismeretlen” nem lesz jó.

Háromértékű logika

	¬
I	*H*
H	*I*
UN	*UN*

∨	*I*	*H*	*UN*
I	*I*	*I*	*I*
H	*I*	*H*	*UN*
UN	*I*	*UN*	*UN*

∧	*I*	*H*	*UN*
I	*I*	*H*	*UN*
H	*H*	*H*	*H*
UN	*UN*	*H*	*UN*

Tehát egy állítás nem csak igaz vagy hamis lehet, hanem „ismeretlen” is és egy WHERE-beli állításnál természetesen csak az számit találatnak, ha igaz az állítás, az „ismeretlen” nem lesz jó.

Furán viselkedik ez a logika:

```
SELECT moziID, filmID
FROM vetít
WHERE idő>”12:00” OR idő <=”12:00”
```

Háromértékű logika

	¬
I	*H*
H	*I*
UN	*UN*

∨	*I*	*H*	*UN*
I	*I*	*I*	*I*
H	*I*	*H*	*UN*
UN	*I*	*UN*	*UN*

∧	*I*	*H*	*UN*
I	*I*	*H*	*UN*
H	*H*	*H*	*H*
UN	*UN*	*H*	*UN*

Tehát egy állítás nem csak igaz vagy hamis lehet, hanem „ismeretlen” is és egy WHERE-beli állításnál természetesen csak az számit találatnak, ha igaz az állítás, az „ismeretlen” nem lesz jó.

Furán viselkedik ez a logika:

SELECT moziID, filmID
FROM vetít
WHERE idő>”12:00” OR idő <=”12:00”

Az lenne jó, ha ez minden filmet felsorol, de sajnos aminek nincs ideje, azt nem sorolja fel.

Háromértékű logika

	$\neg$
I	H
H	I
UN	UN

$\vee$	I	H	UN
I	I	I	I
H	I	H	UN
UN	I	UN	UN

$\wedge$	I	H	UN
I	I	H	UN
H	H	H	H
UN	UN	H	UN

Tehát egy állítás nem csak igaz vagy hamis lehet, hanem „ismeretlen” is és egy WHERE-beli állításnál természetesen csak az számit találatnak, ha igaz az állítás, az „ismeretlen” nem lesz jó.

Furán viselkedik ez a logika:

SELECT moziID, filmID
FROM vetít
WHERE idő>”12:00” OR idő <=”12:00”

Az lenne jó, ha ez minden filmet felsorol, de sajnos aminek nincs ideje, azt nem sorolja fel.

$\Longrightarrow A \vee \neg A \neq I$ a háromértékű logikában, azaz nem teljesül az, amit megszoktunk, hogy vagy az állítás vagy a tagadása igaz lesz.

Háromértékű logika

Hasonlóan: egy mező értéke nem hasonlítható össze a szokásos módon a **NULL** értékkel (mivel **NULL** nem egy konstans).

Háromértékű logika

Hasonlóan: egy mező értéke nem hasonlítható össze a szokásos módon a **NULL** értékkel (mivel **NULL** nem egy konstans).
Erre az **IS NULL**, illetve az **IS NOT NULL** használatosak.

Háromértékű logika

Hasonlóan: egy mező értéke nem hasonlítható össze a szokásos módon a **NULL** értékkel (mivel **NULL** nem egy konstans).
Erre az **IS NULL**, illetve az **IS NOT NULL** használatosak.
Példa 117: Azon filmek címe és rendezője, melyeknek ismerjük a rendezőjét.

```
SELECT cím, rendező FROM film WHERE rendező IS NOT NULL
```

Relációk összekapcsolása (join) SQL2-ben

A következőkben ismertetett nyelvi elemek egy része csak szintaktikai édesítőszer, kifejezhető a
SELECT <attribútumok> FROM R, S WHERE <feltételek> (*)
segítségével.

Relációk összekapcsolása (join) SQL2-ben

A következőkben ismertetett nyelvi elemek egy része csak szintaktikai édesítőszer, kifejezhető a
SELECT <attribútumok> FROM R, S WHERE <feltételek> (*)
segítségével.
Relációk összekapcsolásakor meg kell adni az összekapcsolás módját (belső vagy külső) és a sorok összekapcsolásának feltételét.

Az összekapcsolás módja

- **Belső összekapcsolás** (mint $\bowtie$-nél): **R INNER JOIN S**
  R-nek és S-nek csak azon sorai kerülnek az eredményrelációba, melyekhez van kapcsolódó sor S-ben, illetve R-ben (azaz a másik relációban). (Az, hogy mikor kapcsolódó két sor, az majd a kapcsolódás feltételeinél derül ki.)

Az összekapcsolás módja

- **Belső összekapcsolás** (mint $\bowtie$-nél): **R INNER JOIN S**
  R-nek és S-nek csak azon sorai kerülnek az eredményrelációba, melyekhez van kapcsolódó sor S-ben, illetve R-ben (azaz a másik relációban). (Az, hogy mikor kapcsolódó két sor, az majd a kapcsolódás feltételeinél derül ki.)
- **Bal oldali külső összekapcsolás** (mint $\overset{\supset}{\bowtie}$-nél): **R LEFT [OUTER] JOIN S**
  R-nek azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik S-beli sor.
  Ezekben a csak S-ben szereplő mezők **NULL** értéket kapnak.

Az összekapcsolás módja

- **Belső összekapcsolás** (mint $\bowtie$-nél): **R INNER JOIN S**
  R-nek és S-nek csak azon sorai kerülnek az eredményrelációba, melyekhez van kapcsolódó sor S-ben, illetve R-ben (azaz a másik relációban). (Az, hogy mikor kapcsolódó két sor, az majd a kapcsolódás feltételeinél derül ki.)
- **Bal oldali külső összekapcsolás** (mint $\overset{\supset}{\bowtie}$-nél): **R LEFT [OUTER] JOIN S**
  R-nek azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik S-beli sor.
  Ezekben a csak S-ben szereplő mezők **NULL** értéket kapnak.
- **Jobb oldali külső összekapcsolás** (mint $\overset{\subset}{\bowtie}$-nél): **R RIGHT [OUTER] JOIN S**
  Mint a LEFT OUTER JOIN, de R és S szerepe megcserélődik.
- **Teljes külső összekapcsolás** (mint $\overset{\supset\subset}{\bowtie}$-nél): **R FULL [OUTER] JOIN S**
  Mind R, mind S azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik sor a másik relációból.
  Az ezáltal üresen maradó mezők itt is **NULL** értéket kapnak.

Az összekapcsolás módja

- **Belső összekapcsolás** (mint $\bowtie$-nél): **R INNER JOIN S**
  R-nek és S-nek csak azon sorai kerülnek az eredményrelációba, melyekhez van kapcsolódó sor S-ben, illetve R-ben (azaz a másik relációban). (Az, hogy mikor kapcsolódó két sor, az majd a kapcsolódás feltételeinél derül ki.)
- **Bal oldali külső összekapcsolás** (mint $\overset{\supset}{\bowtie}$-nél): **R LEFT [OUTER] JOIN S**
  R-nek azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik S-beli sor.
  Ezekben a csak S-ben szereplő mezők **NULL** értéket kapnak.
- **Jobb oldali külső összekapcsolás** (mint $\overset{\subset}{\bowtie}$-nél): **R RIGHT [OUTER] JOIN S**
  Mint a LEFT OUTER JOIN, de R és S szerepe megcserélődik.
- **Teljes külső összekapcsolás** (mint $\overset{\supset\subset}{\bowtie}$-nél): **R FULL [OUTER] JOIN S**
  Mind R, mind S azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik sor a másik relációból.
  Az ezáltal üresen maradó mezők itt is **NULL** értéket kapnak.

A direkt szorzat létrehozására az R CROSS JOIN S alak használható, ilyenkor nincs feltétele a sorok összekapcsolásának.
Ez az alapértelmezés, (*) használatakor ilyen illesztés történik.

Az összekapcsolás módja

- **Belső összekapcsolás** (mint $\bowtie$-nél): **R INNER JOIN S**
  R-nek és S-nek csak azon sorai kerülnek az eredményrelációba, melyekhez van kapcsolódó sor S-ben, illetve R-ben (azaz a másik relációban). (Az, hogy mikor kapcsolódó két sor, az majd a kapcsolódás feltételeinél derül ki.)
- **Bal oldali külső összekapcsolás** (mint $\overset{\supset}{\bowtie}$-nél): **R LEFT [OUTER] JOIN S**
  R-nek azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik S-beli sor.
  Ezekben a csak S-ben szereplő mezők **NULL** értéket kapnak.
- **Jobb oldali külső összekapcsolás** (mint $\overset{\subset}{\bowtie}$-nél): **R RIGHT [OUTER] JOIN S**
  Mint a LEFT OUTER JOIN, de R és S szerepe megcserélődik.
- **Teljes külső összekapcsolás** (mint $\overset{\supset\subset}{\bowtie}$-nél): **R FULL [OUTER] JOIN S**
  Mind R, mind S azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik sor a másik relációból.
  Az ezáltal üresen maradó mezők itt is **NULL** értéket kapnak.

A direkt szorzat létrehozására az R CROSS JOIN S alak használható, ilyenkor nincs feltétele a sorok összekapcsolásának.
Ez az alapértelmezés, (*) használatakor ilyen illesztés történik.

A sorok összekapcsolásának feltételei

- **Természetes illesztés:** **R NATURAL [INNER | LEFT | RIGHT | FULL] JOIN S**
  R és S azon sorai illesztődnek, ahol az azonos nevű attribútumok értéke is megegyezik.
  Ez az alapértelmezés.

A sorok összekapcsolásának feltételei

- **Természetes illesztés: R NATURAL [INNER | LEFT | RIGHT | FULL] JOIN S**
  R és S azon sorai illesztődnek, ahol az azonos nevű attribútumok értéke is megegyezik.
  Ez az alapértelmezés.
- **Illesztés azonos nevű attribútumokkal: R [INNER | LEFT | RIGHT | FULL] JOIN S USING (<attribútumok>)**
  R és S azon sorai illesztődnek, ahol az azonos nevű és <attribútumok>-ban felsorolt attribútumok értéke is megegyezik.

A sorok összekapcsolásának feltételei

- **Természetes illesztés: R NATURAL [INNER | LEFT | RIGHT | FULL] JOIN S**
  R és S azon sorai illesztődnek, ahol az azonos nevű attribútumok értéke is megegyezik.
  Ez az alapértelmezés.
- **Illesztés azonos nevű attribútumokkal: R [INNER | LEFT | RIGHT | FULL] JOIN S USING (<attribútumok>)**
  R és S azon sorai illesztődnek, ahol az azonos nevű és <attribútumok>-ban felsorolt attribútumok értéke is megegyezik.
- **Illesztés tetszőleges feltétellel (θ-join): R [INNER | LEFT | RIGHT | FULL] JOIN S ON (<feltétel>)**
  R és S azon sorai illesztődnek, melyek attribútumai eleget tesznek a megadott feltételnek.

Példák relációk összekapcsolására

Példa 118: Kusturica vetített filmjei és vetítési időpontjaik

Példák relációk összekapcsolására

Példa 124: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID

Példák relációk összekapcsolására

Példa 130: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 131: Kusturica összes filmje és vetítési időpontjaik (ha van)

Példák relációk összekapcsolására

Példa 136: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 137: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID

Példák relációk összekapcsolására

Példa 142: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 143: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 144: Vetített filmek címe, rendezője és vetítési időpontjaik

Példák relációk összekapcsolására

Példa 148: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 149: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 150: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít

Példák relációk összekapcsolására

Példa 154: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 155: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 156: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít
Példa 157: ugyanez USING használatával

Példák relációk összekapcsolására

Példa 160: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 161: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 162: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít
Példa 163: ugyanez USING használatával
SELECT cím, rendező, nap, idő FROM film INNER JOIN vetít USING (filmID)

Példák relációk összekapcsolására

Példa 166: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 167: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 168: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít
Példa 169: ugyanez USING használatával
SELECT cím, rendező, nap, idő FROM film INNER JOIN vetít USING (filmID)
Példa 170: Összes film címe, rendezője és vetítési időpontja (ha van)

Példák relációk összekapcsolására

Példa 172: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 173: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 174: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít
Példa 175: ugyanez USING használatával
SELECT cím, rendező, nap, idő FROM film INNER JOIN vetít USING (filmID)
Példa 176: Összes film címe, rendezője és vetítési időpontja (ha van)
SELECT cím, rendező, nap, idő FROM film NATURAL LEFT OUTER JOIN vetít

Példák relációk összekapcsolására

Példa 178: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 179: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 180: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít
Példa 181: ugyanez USING használatával
SELECT cím, rendező, nap, idő FROM film INNER JOIN vetít USING (filmID)
Példa 182: Összes film címe, rendezője és vetítési időpontja (ha van)
SELECT cím, rendező, nap, idő FROM film NATURAL LEFT OUTER JOIN vetít
Példa 183: Az összes film-mozi pár

Példák relációk összekapcsolására

Példa 184: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 185: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
Példa 186: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít
Példa 187: ugyanez USING használatával
SELECT cím, rendező, nap, idő FROM film INNER JOIN vetít USING (filmID)
Példa 188: Összes film címe, rendezője és vetítési időpontja (ha van)
SELECT cím, rendező, nap, idő FROM film NATURAL LEFT OUTER JOIN vetít
Példa 189: Az összes film-mozi pár
SELECT * FROM film CROSS JOIN mozi

DML utasítások — INSERT

Sorokat a relációba az INSERT utasítással szúrhatunk be.

Szintakszis: INSERT INTO <reláció> (<attrib$_1$>, . . . , <attrib$_n$>) VALUES (<érték$_1$>, . . . , <érték$_n$>)

DML utasítások — INSERT

Sorokat a relációba az INSERT utasítással szúrhatunk be.

Szintakszis: INSERT INTO <reláció> (<attrib$_1$>, . . . , <attrib$_n$>) VALUES (<érték$_1$>, . . . , <érték$_n$>)

Hatása: a <reláció> relációba egy új sor kerül, amiben <attrib$_1$> attribútum értéke <érték$_1$>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

DML utasítások — INSERT

Sorokat a relációba az INSERT utasítással szúrhatunk be.

Szintakszis: INSERT INTO <reláció> (<attrib$_1$>, . . . , <attrib$_n$>) VALUES (<érték$_1$>, . . . , <érték$_n$>)

Hatása: a <reláció> relációba egy új sor kerül, amiben <attrib$_1$> attribútum értéke <érték$_1$>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

Példa 192: Egy új film felvétele

INSERT INTO film (cím, rendező) VALUES ('Egy csodálatos elme', 'Ron Howard')

DML utasítások — INSERT

Sorokat a relációba az INSERT utasítással szúrhatunk be.

Szintakszis: INSERT INTO <reláció> (<attrib$_1$>, . . . , <attrib$_n$>) VALUES (<érték$_1$>, . . . , <érték$_n$>)

Hatása: a <reláció> relációba egy új sor kerül, amiben <attrib$_1$> attribútum értéke <érték$_1$>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

Példa 193: Egy új film felvétele

INSERT INTO film (cím, rendező) VALUES ('Egy csodálatos elme', 'Ron Howard')

Megjegyzés:

- a filmID mező az alapértelmezett értékét kapja, de egy trigger (később lesz) segítségével akár automatikusan növekvő számozást is létrehozhatunk.

DML utasítások — INSERT

Sorokat a relációba az INSERT utasítással szúrhatunk be.

Szintakszis: INSERT INTO <reláció> (<attrib$_1$>, . . . , <attrib$_n$>) VALUES (<érték$_1$>, . . . , <érték$_n$>)

Hatása: a <reláció> relációba egy új sor kerül, amiben <attrib$_1$> attribútum értéke <érték$_1$>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

Példa 194: Egy új film felvétele

INSERT INTO film (cím, rendező) VALUES ('Egy csodálatos elme', 'Ron Howard')

Megjegyzés:

- a filmID mező az alapértelmezett értékét kapja, de egy trigger (később lesz) segítségével akár automatikusan növekvő számozást is létrehozhatunk.
- Ha az összes attribútum értékét megadjuk, akkor nem kell őket felsorolni, ebben az esetben a beadott értékek a default attribútumsorrend szerint lesznek hozzárendelve az attribútumokhoz)
- a beszúrt adatokat egy alkérdésből is vehetjük:
  Ha van egy filmregi(filmID, cím, rendező) tábla és azokat az adatokat szeretnénk átvinni a film táblába, amik ott nem szerepelnek:

DML utasítások — INSERT

Sorokat a relációba az INSERT utasítással szúrhatunk be.

Szintakszis: INSERT INTO <reláció> (<attrib$_1$>, . . . , <attrib$_n$>) VALUES (<érték$_1$>, . . . , <érték$_n$>)

Hatása: a <reláció> relációba egy új sor kerül, amiben <attrib$_1$> attribútum értéke <érték$_1$>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

Példa 195: Egy új film felvétele

INSERT INTO film (cím, rendező) VALUES ('Egy csodálatos elme', 'Ron Howard')

Megjegyzés:

- a filmID mező az alapértelmezett értékét kapja, de egy trigger (később lesz) segítségével akár automatikusan növekvő számozást is létrehozhatunk.
- Ha az összes attribútum értékét megadjuk, akkor nem kell őket felsorolni, ebben az esetben a beadott értékek a default attribútumsorrend szerint lesznek hozzárendelve az attribútumokhoz)
- a beszúrt adatokat egy alkérdésből is vehetjük:
  Ha van egy filmregi(filmID, cím, rendező) tábla és azokat az adatokat szeretnénk átvinni a film táblába, amik ott nem szerepelnek:

```
INSERT INTO film
        SELECT filmregi.filmID, filmregi.cím, filmregi.rendező
        FROM filmregi
        WHERE filmregi.filmID NOT IN
                (SELECT filmID FROM film)
```

DML utasítások — UPDATE

Sorokat a relációban az UPDATE utasítással módosíthatunk.
Szintakszis: UPDATE <reláció> SET <attrib$_1$>=<érték$_1$>, . . . , <attrib$_n$>=<érték$_n$>
WHERE <feltétel>

DML utasítások — UPDATE

Sorokat a relációban az UPDATE utasítással módosíthatunk.
Szintakszis: UPDATE <reláció> SET <attrib$_1$>=<érték$_1$>, . . . , <attrib$_n$>=<érték$_n$> WHERE <feltétel>
Hatása: a <reláció> reláció minden sorában, amelyik illeszkedik a <feltétel> feltételre <attrib$_i$> értéke <érték$_i$> lesz.

DML utasítások — UPDATE

Sorokat a relációban az UPDATE utasítással módosíthatunk.
Szintakszis: UPDATE <reláció> SET <attrib$_1$>=<érték$_1$>, . . . , <attrib$_n$>=<érték$_n$> WHERE <feltétel>
Hatása: a <reláció> reláció minden sorában, amelyik illeszkedik a <feltétel> feltételre <attrib$_i$> értéke <érték$_i$> lesz.
Példa 198: Az előbb beszúrt rendező nevének átírása rövidített alakba

DML utasítások — UPDATE

Sorokat a relációban az UPDATE utasítással módosíthatunk.
Szintakszis: UPDATE <reláció> SET <attrib$_1$>=<érték$_1$>, . . . , <attrib$_n$>=<érték$_n$> WHERE <feltétel>
Hatása: a <reláció> reláció minden sorában, amelyik illeszkedik a <feltétel> feltételre <attrib$_i$> értéke <érték$_i$> lesz.
Példa 199: Az előbb beszúrt rendező nevének átírása rövidített alakba
UPDATE film SET rendező='R. Howard' WHERE rendező='Ron Howard'

DML utasítások — DELETE

Sorokat egy relációból a DELETE utasítással törölhetünk.
Szintakszis: DELETE FROM <reláció> WHERE <feltétel>

DML utasítások — DELETE

Sorokat egy relációból a DELETE utasítással törölhetünk.
Szintakszis: DELETE FROM <reláció> WHERE <feltétel>
Hatása: a <reláció> reláció feltételre illeszkedő sorait törli.

DML utasítások — DELETE

Sorokat egy relációból a DELETE utasítással törölhetünk.
Szintakszis: DELETE FROM <reláció> WHERE <feltétel>
Hatása: a <reláció> reláció feltételre illeszkedő sorait törli.
Megjegyzés: a WHERE <feltétel> rész elhagyása esetén a reláció összes sorát törli.

DML utasítások — DELETE

Sorokat egy relációból a DELETE utasítással törölhetünk.
Szintakszis: DELETE FROM <reláció> WHERE <feltétel>
Hatása: a <reláció> reláció feltételre illeszkedő sorait törli.
Megjegyzés: a WHERE <feltétel> rész elhagyása esetén a reláció összes sorát törli.
Példa 203: Azon filmek törlése, amiknek a rendezője E. K. monogrammú
DELETE FROM film WHERE rendező LIKE 'E.% K%'

SQL Data Definition Language

- Séma létrehozása
- Séma törlése
- Séma módosítása

SQL Data Definition Language

- Séma létrehozása
- Séma törlése
- Séma módosítása
- Indexek létrehozása és kezelése (az indexekről később lesz szó részletesen)

SQL Data Definition Language

- Séma létrehozása
- Séma törlése
- Séma módosítása
- Indexek létrehozása és kezelése (az indexekről később lesz szó részletesen)
- Nézetek létrehozása

SQL Data Definition Language

- Séma létrehozása
- Séma törlése
- Séma módosítása
- Indexek létrehozása és kezelése (az indexekről később lesz szó részletesen)
- Nézetek létrehozása
- Kényszerek létrehozása

SQL Data Definition Language

- Séma létrehozása
- Séma törlése
- Séma módosítása
- Indexek létrehozása és kezelése (az indexekről később lesz szó részletesen)
- Nézetek létrehozása
- Kényszerek létrehozása
- Triggerek (kicsit)

A triggerek már az SQL3-hoz tartoznak, a triggerek segítségével az adatbázis valamely változásakor egy tárolt eljárást hajthatunk végre. Itt fogunk beszélni a rekurzióról is, mert az is SQL3-as dolog, de az lekérdezés és nem DDL, a segítségével egy reláció tranzitív lezárását lehet kiszámolni rekurzívan.

Relációk létrehozása

Relációk létrehozására a **CREATE TABLE** használandó.
Minden attribútumnak típust és a reláción belül egyedi nevet kell adni.

Relációk létrehozása

Relációk létrehozására a **CREATE TABLE** használandó.
Minden attribútumnak típust és a reláción belül egyedi nevet kell adni.
Szükség esetén megadható alapértelmezett érték is (**DEFAULT** kulcsszóval), melyeket az attribútum azon sorokban vesz fel, ahol a beszúráskor nem adtuk meg a konkrét értékét.

Relációk létrehozása

Relációk létrehozására a **CREATE TABLE** használandó.
Minden attribútumnak típust és a reláción belül egyedi nevet kell adni.
Szükség esetén megadható alapértelmezett érték is (**DEFAULT** kulcsszóval), melyeket az attribútum azon sorokban vesz fel, ahol a beszúráskor nem adtuk meg a konkrét értékét.
Lehetőség van arra is, hogy kényszereket definiáljunk az attribútumokra (pl. attribútum nem **NULL**, elsődleges kulcs, idegen kulcs, stb.), ezekről később lesz szó.

Relációk létrehozása

Relációk létrehozására a **CREATE TABLE** használandó.
Minden attribútumnak típust és a reláción belül egyedi nevet kell adni.
Szükség esetén megadható alapértelmezett érték is (**DEFAULT** kulcsszóval), melyeket az attribútum azon sorokban vesz fel, ahol a beszúráskor nem adtuk meg a konkrét értékét.
Lehetőség van arra is, hogy kényszereket definiáljunk az attribútumokra (pl. attribútum nem **NULL**, elsődleges kulcs, idegen kulcs, stb.), ezekről később lesz szó.
Szintakszis:
CREATE TABLE <relációnév> (
<attrib$_1$> <adattípus$_1$> [DEFAULT <érték>], . . .
<attrib$_n$> <adattípus$_n$> [DEFAULT <érték>]
)

Példa 204: Film reláció létrehozása

```
CREATE TABLE film(
   filmID number(5),
   cím varchar(50),
   rendező char(30),
   év number(4),
   hossz number(3) DEFAULT 90)
```

Példa 205: Film reláció létrehozása

```
CREATE TABLE film(
   filmID number(5),
   cím varchar(50),
   rendező char(30),
   év number(4),
   hossz number(3) DEFAULT 90)
```

A lehetséges adattípusok függenek az adatbáziskezelőtől.

Példa 206: Film reláció létrehozása

```
CREATE TABLE film(
   filmID number(5),
   cím varchar(50),
   rendező char(30),
   év number(4),
   hossz number(3) DEFAULT 90)
```

A lehetséges adattípusok függenek az adatbáziskezelőtől.

A főbb típusok:

char(n)	*n* hosszú karaktersorozat
varchar(n)	maximum *n* hosszú karaktersorozat
number(n,m)	*n* hosszú, *m* tizedesjegyű szám
date	dátum

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>
Példa 211: Film reláció törlése :-(
DROP TABLE film

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>
Példa 215: Film reláció törlése :-(
DROP TABLE film
Relációk módosítására az **ALTER TABLE** használandó.

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>
Példa 219: Film reláció törlése :-(
DROP TABLE film
Relációk módosítására az **ALTER TABLE** használandó.
Lehetőség van új attribútum definiálására (**ADD**), attribútum törlésére (**DROP**), attribútum adattípusának módosítására (**MODIFY**), kényszerek módosítására, alapértelmezett érték megadására.
Mindezek csak a jelenlegi adatokkal konzisztensen végezhetők el.
Például nem törölhető olyan attribútum, amire még van hivatkozás.

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>
Példa 223: Film reláció törlése :-(
DROP TABLE film
Relációk módosítására az **ALTER TABLE** használandó.
Lehetőség van új attribútum definiálására (**ADD**), attribútum törlésére (**DROP**), attribútum adattípusának módosítására (**MODIFY**), kényszerek módosítására, alapértelmezett érték megadására.
Mindezek csak a jelenlegi adatokkal konzisztensen végezhetők el.
Például nem törölhető olyan attribútum, amire még van hivatkozás.
Szintaxis: ALTER TABLE <relációnév> <opciók>

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>
Példa 227: Film reláció törlése :-(
DROP TABLE film
Relációk módosítására az **ALTER TABLE** használandó.
Lehetőség van új attribútum definiálására (**ADD**), attribútum törlésére (**DROP**), attribútum adattípusának módosítására (**MODIFY**), kényszerek módosítására, alapértelmezett érték megadására.
Mindezek csak a jelenlegi adatokkal konzisztensen végezhetők el.
Például nem törölhető olyan attribútum, amire még van hivatkozás.
Szintaxis: ALTER TABLE <relációnév> <opciók>
Példa 228: Vetít relációhoz helyár hozzáadása
ALTER TABLE vetít ADD helyár number(4)

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>
Példa 231: Film reláció törlése :-(
DROP TABLE film
Relációk módosítására az **ALTER TABLE** használandó.
Lehetőség van új attribútum definiálására (**ADD**), attribútum törlésére (**DROP**), attribútum adattípusának módosítására (**MODIFY**), kényszerek módosítására, alapértelmezett érték megadására.
Mindezek csak a jelenlegi adatokkal konzisztensen végezhetők el.
Például nem törölhető olyan attribútum, amire még van hivatkozás.
Szintaxis: ALTER TABLE <relációnév> <opciók>
Példa 232: Vetít relációhoz helyár hozzáadása
ALTER TABLE vetít ADD helyár number(4)
Példa 233: Mozi relációból a város eltávolítása
ALTER TABLE mozi DROP város

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.
Szintaxis: DROP TABLE <relációnév>
Példa 235: Film reláció törlése :-(
DROP TABLE film
Relációk módosítására az **ALTER TABLE** használandó.
Lehetőség van új attribútum definiálására (**ADD**), attribútum törlésére (**DROP**), attribútum adattípusának módosítására (**MODIFY**), kényszerek módosítására, alapértelmezett érték megadására.
Mindezek csak a jelenlegi adatokkal konzisztensen végezhetők el.
Például nem törölhető olyan attribútum, amire még van hivatkozás.
Szintaxis: ALTER TABLE <relációnév> <opciók>
Példa 236: Vetít relációhoz helyár hozzáadása
ALTER TABLE vetít ADD helyár number(4)
Példa 237: Mozi relációból a város eltávolítása
ALTER TABLE mozi DROP város
Példa 238: Ha a helyárakat ezentúl dollárban számoljuk . . .
ALTER TABLE vetít MODIFY helyár number(4,2)

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.
SQL2-nek sem része, de gyakori, ezért tanuljuk.
Szintaxis: CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.
SQL2-nek sem része, de gyakori, ezért tanuljuk.
Szintaxis: CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)

Példa 241: index a filmcím, rendező párra
CREATE INDEX cím-rend ON film(cím, rendező)

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.
SQL2-nek sem része, de gyakori, ezért tanuljuk.
Szintaxis: CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)

Példa 242: index a filmcím, rendező párra
CREATE INDEX cím-rend ON film(cím, rendező)

Ha meguntuk, el lehet dobni: DROP INDEX cím-rend

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.
SQL2-nek sem része, de gyakori, ezért tanuljuk.
Szintaxis: CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)

Példa 243: index a filmcím, rendező párra
CREATE INDEX cím-rend ON film(cím, rendező)

Ha meguntuk, el lehet dobni: DROP INDEX cím-rend

- előny: gyors keresés lehetséges az index segítségével

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.
SQL2-nek sem része, de gyakori, ezért tanuljuk.
Szintaxis: CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)

Példa 244: index a filmcím, rendező párra
CREATE INDEX cím-rend ON film(cím, rendező)

Ha meguntuk, el lehet dobni: DROP INDEX cím-rend

- előny: gyors keresés lehetséges az index segítségével
- hátrány: az adatszerkezetet karban kell tartani, így lassítja a beszúrást, törlést, módosítást

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.
SQL2-nek sem része, de gyakori, ezért tanuljuk.
Szintaxis: CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)

Példa 245: index a filmcím, rendező párra
CREATE INDEX cím-rend ON film(cím, rendező)

Ha meguntuk, el lehet dobni: DROP INDEX cím-rend

- előny: gyors keresés lehetséges az index segítségével
- hátrány: az adatszerkezetet karban kell tartani, így lassítja a beszúrást, törlést, módosítást
- az a fontos, hogy miből van több, módosításból vagy lekérdezésből?

Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.
SQL2-nek sem része, de gyakori, ezért tanuljuk.
Szintaxis: CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)

Példa 246: index a filmcím, rendező párra
CREATE INDEX cím-rend ON film(cím, rendező)

Ha meguntuk, el lehet dobni: DROP INDEX cím-rend

- előny: gyors keresés lehetséges az index segítségével
- hátrány: az adatszerkezetet karban kell tartani, így lassítja a beszúrást, törlést, módosítást
- az a fontos, hogy miből van több, módosításból vagy lekérdezésből?
- néha a rendszer magától létrehoz indexet (lásd kulcsok)

Adatbázisok elmélete

SQL nézetek, kényszerek, triggerek

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

11. előadás

Nézetek létrehozása

Permanensen létező, származtatott relációt hoz létre, amire hivatkozhatunk lekérdezésekkor is.

Nézetek létrehozása

Permanensen létező, származtatott relációt hoz létre, amire hivatkozhatunk lekérdezésekkor is.

Szintaxis: CREATE VIEW <új reláció neve> AS <lekérdezés>

Nézetek létrehozása

Permanensen létező, származtatott relációt hoz létre, amire hivatkozhatunk lekérdezésekkor is.

Szintaxis: CREATE VIEW <új reláció neve> AS <lekérdezés>

Nézetek létrehozása

Példa 247: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

Nézetek létrehozása

Példa 249: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik

Nézetek létrehozása

Példa 251: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)

Nézetek létrehozása

Példa 253: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)
- de változtathatók az adatok korlátozottan a nézettáblán keresztül is
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk:

Nézetek létrehozása

Példa 255: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)
- de változtathatók az adatok korlátozottan a nézettáblán keresztül is
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk:
  Példa 256: Milyen Almodovar filmeket vetítenek most?
  `SELECT cím FROM Almodovarfilm NATURAL INNER JOIN vetít`

Nézetek létrehozása

Példa 257: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)
- de változtathatók az adatok korlátozottan a nézettáblán keresztül is
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk:
  Példa 258: Milyen Almodovar filmeket vetítenek most?
  `SELECT cím FROM Almodovarfilm NATURAL INNER JOIN vetít`
- Egy ilyen kérdés kiértékelésekor az Almodovarfilm nézettábla helyére a lekérdezésfeldolgozó berakja az őt definiáló SELECT-et

Nézetek létrehozása

Példa 259: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)
- de változtathatók az adatok korlátozottan a nézettáblán keresztül is
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk:
  Példa 260: Milyen Almodovar filmeket vetítenek most?
  `SELECT cím FROM Almodovarfilm NATURAL INNER JOIN vetít`
- Egy ilyen kérdés kiértékelésekor az Almodovarfilm nézettábla helyére a lekérdezésfeldolgozó berakja az őt definiáló SELECT-et
- Lehet új attribútumnevet adni a nézettáblában

Nézetek létrehozása

Példa 261: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)
- de változtathatók az adatok korlátozottan a nézettáblán keresztül is
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk:
  Példa 262: Milyen Almodovar filmeket vetítenek most?
  `SELECT cím FROM Almodovarfilm NATURAL INNER JOIN vetít`
- Egy ilyen kérdés kiértékelésekor az Almodovarfilm nézettábla helyére a lekérdezésfeldolgozó berakja az őt definiáló SELECT-et
- Lehet új attribútumnevet adni a nézettáblában

Megszüntetése: `DROP VIEW Almodovarfilm`

Nézetek létrehozása

Példa 263: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
    SELECT filmID, cím
    FROM film
    WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)
- de változtathatók az adatok korlátozottan a nézettáblán keresztül is
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk:
  Példa 264: Milyen Almodovar filmeket vetítenek most?
  `SELECT cím FROM Almodovarfilm NATURAL INNER JOIN vetít`
- Egy ilyen kérdés kiértékelésekor az Almodovarfilm nézettábla helyére a lekérdezésfeldolgozó berakja az őt definiáló SELECT-et
- Lehet új attribútumnevet adni a nézettáblában

Megszüntetése: `DROP VIEW Almodovarfilm`

Ezután már nem lehet olyan lekérdezést írni, amiben ez szerepel.

Kényszerek

Kényszerek csoportosítása

Kényszer típusa szerint

- Elsődleges kulcs (**PRIMARY KEY**)
- Egyértékűségi megszorítások (**UNIQUE**)
- Hivatkozási épség, idegen kulcs (**FOREIGN KEY**)
- NULL érték tiltása (**NOT NULL**)

Kényszerek

Kényszerek csoportosítása

Kényszer típusa szerint

- Elsődleges kulcs (**PRIMARY KEY**)
- Egyértékűségi megszorítások (**UNIQUE**)
- Hivatkozási épség, idegen kulcs (**FOREIGN KEY**)
- NULL érték tiltása (**NOT NULL**)
- Értékkészlet (**CHECK**)
  - attribútumra vonatkozó feltétel
  - sorra vonatkozó feltétel
  - globális feltétel

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**
Attribútum(ok) egyediségének megkövetelése: **UNIQUE**

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**
Attribútum(ok) egyediségének megkövetelése: **UNIQUE**
Mindkét esetben az adott attribútumoknak egyértelműen azonosítaniuk kell a sort.

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**
Attribútum(ok) egyediségének megkövetelése: **UNIQUE**
Mindkét esetben az adott attribútumoknak egyértelműen azonosítaniuk kell a sort.
Különbség: PRIMARY KEY csak egy lehet, idegen kulcs csak erre hivatkozhat, sok rendszer automatikusan indexet hoz rá létre.

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**
Attribútum(ok) egyediségének megkövetelése: **UNIQUE**
Mindkét esetben az adott attribútumoknak egyértelműen azonosítaniuk kell a sort.
Különbség: PRIMARY KEY csak egy lehet, idegen kulcs csak erre hivatkozhat, sok rendszer automatikusan indexet hoz rá létre.
Szintaxis:

a tábla létrehozásakor, az attribútum definíciójában:

<attribútum> <típus> { PRIMARY KEY | UNIQUE }

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**
Attribútum(ok) egyediségének megkövetelése: **UNIQUE**
Mindkét esetben az adott attribútumoknak egyértelműen azonosítaniuk kell a sort.
Különbség: PRIMARY KEY csak egy lehet, idegen kulcs csak erre hivatkozhat, sok rendszer automatikusan indexet hoz rá létre.
Szintaxis:

a tábla létrehozásakor, az attribútum definíciójában:
<attribútum> <típus> { PRIMARY KEY | UNIQUE }

relációdefiníción belül, önállóan, külön sorban:
{ PRIMARY KEY | UNIQUE } (<attrib$_1$>,. . . ,<attrib$_k$>)

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**
Attribútum(ok) egyediségének megkövetelése: **UNIQUE**
Mindkét esetben az adott attribútumoknak egyértelműen azonosítaniuk kell a sort.
Különbség: PRIMARY KEY csak egy lehet, idegen kulcs csak erre hivatkozhat, sok rendszer automatikusan indexet hoz rá létre.
Szintaxis:

a tábla létrehozásakor, az attribútum definíciójában:
<attribútum> <típus> { PRIMARY KEY | UNIQUE }

relációdefiníción belül, önállóan, külön sorban:
{ PRIMARY KEY | UNIQUE } (<attrib$_1$>,. . . ,<attrib$_k$>)

Ilyenkor (<attrib$_1$>, . . . ,<attrib$_k$>) együtt a kulcs. Ha egy kulcs több attribútumból áll, akkor csak így lehet megadni.

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**
Attribútum(ok) egyediségének megkövetelése: **UNIQUE**
Mindkét esetben az adott attribútumoknak egyértelműen azonosítaniuk kell a sort.
Különbség: PRIMARY KEY csak egy lehet, idegen kulcs csak erre hivatkozhat, sok rendszer automatikusan indexet hoz rá létre.
Szintaxis:

a tábla létrehozásakor, az attribútum definíciójában:
<attribútum> <típus> { PRIMARY KEY | UNIQUE }

relációdefiníción belül, önállóan, külön sorban:
{ PRIMARY KEY | UNIQUE } (<attrib$_1$>,. . . ,<attrib$_k$>)

Ilyenkor (<attrib$_1$>, . . . ,<attrib$_k$>) együtt a kulcs. Ha egy kulcs több attribútumból áll, akkor csak így lehet megadni.
A kulcsfeltételeket a rendszer minden beszúrás és módosítás előtt ellenőrzi, ezért van automatikusan index rájuk. És persze emiatt óvatosan kell a kulcsok megadásával bánni, mert nagyon lelassíthatják az adatmódosításokat.

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.
Másik reláció elsődleges kulcsára hivatkozás. Kulcsszavak: **FOREIGN KEY, REFERENCES**

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.
Másik reláció elsődleges kulcsára hivatkozás. Kulcsszavak: **FOREIGN KEY, REFERENCES**
Szintaxis:

attribútum definíciójában:

<attribútum> <típus> REFERENCES <hivatkozott reláció>(<hivatkozott attribútum>)

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.
Másik reláció elsődleges kulcsára hivatkozás. Kulcsszavak: **FOREIGN KEY, REFERENCES**
Szintaxis:

attribútum definíciójában:

<attribútum> <típus> REFERENCES <hivatkozott reláció>(<hivatkozott attribútum>)

relációdefiníción belül, önállóan, külön sorban:

FOREIGN KEY <attribútumok>
REFERENCES <hivatkozott reláció>(<hivatkozott attribútumok>)

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.
Másik reláció elsődleges kulcsára hivatkozás. Kulcsszavak: **FOREIGN KEY, REFERENCES**
Szintaxis:

attribútum definíciójában:

<attribútum> <típus> REFERENCES <hivatkozott reláció>(<hivatkozott attribútum>)

relációdefiníción belül, önállóan, külön sorban:

FOREIGN KEY <attribútumok>
REFERENCES <hivatkozott reláció>(<hivatkozott attribútumok>)

A **FOREIGN KEY** kulcsszó után álló attribútumokat nevezzük **idegen kulcsoknak**.
A fenti deklaráció jelentése: ha létezik egy sor a relációban, ahol az idegen kulcsban levő attribútumok valami adott értékeket vesznek fel, akkor léteznie kell a hivatkozott relációban is egy olyan sornak, ahol a hivatkozott attribútumok értékei ugyanezek.

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.
Másik reláció elsődleges kulcsára hivatkozás. Kulcsszavak: **FOREIGN KEY, REFERENCES**
Szintaxis:

attribútum definíciójában:

<attribútum> <típus> REFERENCES <hivatkozott reláció>(<hivatkozott attribútum>)

relációdefiníción belül, önállóan, külön sorban:

FOREIGN KEY <attribútumok>
REFERENCES <hivatkozott reláció>(<hivatkozott attribútumok>)

A **FOREIGN KEY** kulcsszó után álló attribútumokat nevezzük **idegen kulcsoknak**.
A fenti deklaráció jelentése: ha létezik egy sor a relációban, ahol az idegen kulcsban levő attribútumok valami adott értékeket vesznek fel, akkor léteznie kell a hivatkozott relációban is egy olyan sornak, ahol a hivatkozott attribútumok értékei ugyanezek.
Kell, hogy a hivatkozott attribútumok elsődleges kulcsot alkossanak a hivatkozott relációban.

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.
Másik reláció elsődleges kulcsára hivatkozás. Kulcsszavak: **FOREIGN KEY, REFERENCES**
Szintaxis:

attribútum definíciójában:

<attribútum> <típus> REFERENCES <hivatkozott reláció>(<hivatkozott attribútum>)

relációdefiníción belül, önállóan, külön sorban:

FOREIGN KEY <attribútumok>
REFERENCES <hivatkozott reláció>(<hivatkozott attribútumok>)

A **FOREIGN KEY** kulcsszó után álló attribútumokat nevezzük **idegen kulcsoknak**.
A fenti deklaráció jelentése: ha létezik egy sor a relációban, ahol az idegen kulcsban levő attribútumok valami adott értékeket vesznek fel, akkor léteznie kell a hivatkozott relációban is egy olyan sornak, ahol a hivatkozott attribútumok értékei ugyanezek.
Kell, hogy a hivatkozott attribútumok elsődleges kulcsot alkossanak a hivatkozott relációban.
Az idegen kulcs deklarálása után záradékban megadható, mi történjen, ha a hivatkozott mező megváltozik, törlődik, illetve ha a hivatkozó mező megváltozna.
Lehetőség van a változás/törlés megtiltására vagy a hivatkozó mező kijavítására is.

NULLitás

A **NOT NULL** kulcsszóval megtilthatjuk egy attribútum esetében a **NULL** (ismeretlen, nem létező) érték megadását.
Ezt használva mindenképpen valamilyen érték kerül az attribútum valamennyi sorába, ezért csak kötelezően megadandó attribútumok esetén használjuk!

NULLitás

A **NOT NULL** kulcsszóval megtilthatjuk egy attribútum esetében a **NULL** (ismeretlen, nem létező) érték megadását.
Ezt használva mindenképpen valamilyen érték kerül az attribútum valamennyi sorába, ezért csak kötelezően megadandó attribútumok esetén használjuk!
Szintaxis: az attribútum definíciójában:
<attribútum> <típus> NOT NULL

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)
sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)
sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)
több relációra vonatkozó globális feltétel:

CREATE ASSERTION <kényszernév> CHECK(<feltétel>)

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.

Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)

sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)

több relációra vonatkozó globális feltétel:

CREATE ASSERTION <kényszernév> CHECK(<feltétel>)

Tipikus attribútumra vonatkozó feltételek lehetnek:

értékkészlet felsorolása: <attribútum> IN (<érték1>, . . . , <értékn>)

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.

Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)

sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)

több relációra vonatkozó globális feltétel:

CREATE ASSERTION <kényszernév> CHECK(<feltétel>)

Tipikus attribútumra vonatkozó feltételek lehetnek:

értékkészlet felsorolása: <attribútum> IN (<érték1>, . . . , <értékn>)

intervallum megadása: <attribútum> BETWEEN <alsó határ> AND <felső határ>

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)
sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)
több relációra vonatkozó globális feltétel:

CREATE ASSERTION <kényszernév> CHECK(<feltétel>)

Tipikus attribútumra vonatkozó feltételek lehetnek:

értékkészlet felsorolása: <attribútum> IN (<érték1>, . . . , <értékn>)
intervallum megadása: <attribútum> BETWEEN <alsó határ> AND <felső határ>

De bármi állhat itt, ami WHERE után szerepelhet, akár alkérdés is.

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)
sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)
több relációra vonatkozó globális feltétel:

CREATE ASSERTION <kényszernév> CHECK(<feltétel>)

Tipikus attribútumra vonatkozó feltételek lehetnek:

értékkészlet felsorolása: <attribútum> IN (<érték1>, . . . , <értékn>)
intervallum megadása: <attribútum> BETWEEN <alsó határ> AND <felső határ>

De bármi állhat itt, ami WHERE után szerepelhet, akár alkérdés is.
Például a vetít tábla létrehozásakor beírhatunk egy ilyen sort:
CHECK (filmID IN (SELECT film.filmID FROM film))

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)
sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)
több relációra vonatkozó globális feltétel:
CREATE ASSERTION <kényszernév> CHECK(<feltétel>)

Tipikus attribútumra vonatkozó feltételek lehetnek:

értékkészlet felsorolása: <attribútum> IN (<érték1>, . . . , <értékn>)
intervallum megadása: <attribútum> BETWEEN <alsó határ> AND <felső határ>

De bármi állhat itt, ami WHERE után szerepelhet, akár alkérdés is.
Például a vetít tábla létrehozásakor beírhatunk egy ilyen sort:
CHECK (filmID IN (SELECT film.filmID FROM film))

Ebben az esetben a vetít tábla minden egyes változásakor leellenőrizzük, hogy létezik-e a megfelelő film a film táblában.

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
Szintaxis:

attribútumra vonatkozó feltétel: <attribútum> <típus> CHECK (<feltétel>)
sorra vonatkozó feltétel, relációdefinícióban: CHECK (<feltétel>)
több relációra vonatkozó globális feltétel:
CREATE ASSERTION <kényszernév> CHECK(<feltétel>)

Tipikus attribútumra vonatkozó feltételek lehetnek:

értékkészlet felsorolása: <attribútum> IN (<érték1>, . . . , <értékn>)
intervallum megadása: <attribútum> BETWEEN <alsó határ> AND <felső határ>

De bármi állhat itt, ami WHERE után szerepelhet, akár alkérdés is.
Például a vetít tábla létrehozásakor beírhatunk egy ilyen sort:
CHECK (filmID IN (SELECT film.filmID FROM film))

Ebben az esetben a vetít tábla minden egyes változásakor leellenőrizzük, hogy létezik-e a megfelelő film a film táblában.

Baj ezzel: csak akkor ellenőrzi, ha a vetít táblával történik valami, azt simán hagyja, hogy a film táblából töröljek, pedig ilyenkor is elromolhat.

Erre megoldás az **ASSERTION**:

```
CREATE ASSERTION vetít-film CHECK (
                    vetít.filmID IN (SELECT film.filmID FROM film) )
```

Erre megoldás az **ASSERTION**:

```
CREATE ASSERTION vetít-film CHECK (
                    vetít.filmID IN (SELECT film.filmID FROM film) )
```

Ezt a rendszer minden olyan alkalommal ellenőrzi, ha vagy a vetít vagy a film változik.

Erre megoldás az **ASSERTION**:

```
CREATE ASSERTION vetít-film CHECK (
                    vetít.filmID IN (SELECT film.filmID FROM film) )
```

Ezt a rendszer minden olyan alkalommal ellenőrzi, ha vagy a vetít vagy a film változik.

Megjegyzések:

a kényszerek a **CONSTRAINT** kulcsszó segítségével elnevezhetőek (a PRIMARY KEY, CHECK elé írva)

Erre megoldás az **ASSERTION**:

```
CREATE ASSERTION vetít-film CHECK (
                    vetít.filmID IN (SELECT film.filmID FROM film) )
```

Ezt a rendszer minden olyan alkalommal ellenőrzi, ha vagy a vetít vagy a film változik.

Megjegyzések:

a kényszerek a **CONSTRAINT** kulcsszó segítségével elnevezhetőek (a PRIMARY KEY, CHECK elé írva)

új kényszer hozzáadására, meglévő törlésére az ALTER TABLE . . . {ADD | DROP} CONSTRAINT ad lehetőséget.

Példák kényszerekre

A film és a vetít relációk kényszerekkel kiegészített létrehozása:

```
CREATE TABLE film(
   filmID number(5) PRIMARY KEY,
   cím varchar(50) NOT NULL,
   rendező char(30) NOT NULL,
   év number(4) CHECK (év >= 1900),
   hossz number(3) DEFAULT 90 CHECK (hossz BETWEEN 1 AND 300),
   szinkronizált char(1) DEFAULT 'N' CHECK (szinkronizált IN ('I','N')),
   UNIQUE(cím, rendező)
)
```

Példák kényszerekre

A film és a vetít relációk kényszerekkel kiegészített létrehozása:

```
CREATE TABLE film(
   filmID number(5) PRIMARY KEY,
   cím varchar(50) NOT NULL,
   rendező char(30) NOT NULL,
   év number(4) CHECK (év >= 1900),
   hossz number(3) DEFAULT 90 CHECK (hossz BETWEEN 1 AND 300),
   szinkronizált char(1) DEFAULT 'N' CHECK (szinkronizált IN ('I','N')),
   UNIQUE(cím, rendező)
)
CREATE TABLE vetít(
   filmID number(5) REFERENCES film(filmID),
   moziID number(3) REFERENCES mozi(moziID),
   nap char(9),
   idő char(5) NOT NULL,
   CHECK (nap IN ('hétfő', 'kedd', 'szerda', 'csütörtök', 'péntek', 'szombat', 'vasárnap'))
)
```

Triggerek

Triggerek

SQL2: mindenféle, elég összetett CHECK feltételek, de a rendszerbe bele van építve, hogy mikor kell ellenőriznie valami feltételt

Triggerek

SQL2: mindenféle, elég összetett CHECK feltételek, de a rendszerbe bele van építve, hogy mikor kell ellenőriznie valami feltételt

SQL3-as szemlélet: lehetőség van arra, hogy mi mondjuk meg, mikor legyen ellenőrzés

Trigger:

- Mikor legyen ellenőrzés (adott relációba való beszúráskor, törléskor, módosításkor, tranzakció végén)

Triggerek

SQL2: mindenféle, elég összetett CHECK feltételek, de a rendszerbe bele van építve, hogy mikor kell ellenőriznie valami feltételt

SQL3-as szemlélet: lehetőség van arra, hogy mi mondjuk meg, mikor legyen ellenőrzés

Trigger:

- Mikor legyen ellenőrzés (adott relációba való beszúráskor, törléskor, módosításkor, tranzakció végén)
- Mi legyen a feltétel, amit ekkor ellenőrzünk?

Triggerek

SQL2: mindenféle, elég összetett CHECK feltételek, de a rendszerbe bele van építve, hogy mikor kell ellenőriznie valami feltételt

SQL3-as szemlélet: lehetőség van arra, hogy mi mondjuk meg, mikor legyen ellenőrzés

Trigger:

- Mikor legyen ellenőrzés (adott relációba való beszúráskor, törléskor, módosításkor, tranzakció végén)
- Mi legyen a feltétel, amit ekkor ellenőrzünk?
- Ha a feltétel teljesül, akkor mit csináljunk? (akadályozzunk meg valamit, csináljunk vissza valamit, vagy bármi más)

Triggerek

SQL2: mindenféle, elég összetett CHECK feltételek, de a rendszerbe bele van építve, hogy mikor kell ellenőriznie valami feltételt

SQL3-as szemlélet: lehetőség van arra, hogy mi mondjuk meg, mikor legyen ellenőrzés

Trigger:

- Mikor legyen ellenőrzés (adott relációba való beszúráskor, törléskor, módosításkor, tranzakció végén)
- Mi legyen a feltétel, amit ekkor ellenőrzünk?
- Ha a feltétel teljesül, akkor mit csináljunk? (akadályozzunk meg valamit, csináljunk vissza valamit, vagy bármi más)

Paraméternek adható meg, hogy a kiváltó esemény előtt/helyett/után történjen a cselekvés és még sok más is.

Példa triggerre

Séma: GyártásIrányító(név, cím, azonosító, nettóBevétel)

Példa triggerre

Séma: GyártásIrányító(név, cím, azonosító, nettóBevétel)

```
CREATE TRIGGER NetBevétTrigger
AFTER UPDATE OF nettóBevétel ON GyártásIrányító
REFERENCING
OLD AS RégiSor
NEW AS Újsor
WHEN (RégiSor.nettóBevétel > Újsor.nettóBevétel)
SET nettóBevétel=Régisor.nettóBevétel
WHERE azonosító=Újsor.azonosító
FOR EACH ROW
```

Példa triggerre

Séma: GyártásIrányító(név, cím, azonosító, nettóBevétel)

```
CREATE TRIGGER NetBevétTrigger
AFTER UPDATE OF nettóBevétel ON GyártásIrányító
REFERENCING
OLD AS RégiSor
NEW AS Újsor
WHEN (RégiSor.nettóBevétel > Újsor.nettóBevétel)
SET nettóBevétel=Régisor.nettóBevétel
WHERE azonosító=Újsor.azonosító
FOR EACH ROW
```

$\Longrightarrow$ Ha valakinek csökkenne a bevétele, nem hagyjuk!

Rekurzió

SQL3-as dolog, ideiglenes elképzelés

Rekurzió

SQL3-as dolog, ideiglenes elképzelés

Lekérdezés és nem DDL (csak úgy kerül ide, hogy ez is SQL3)

Rekurzió

SQL3-as dolog, ideiglenes elképzelés

Lekérdezés és nem DDL (csak úgy kerül ide, hogy ez is SQL3)

Példa: Van egy Járat(honnan, hova) táblánk, amiben azt tároljuk, hogy mely városokból hova mennek közvetlenül gépek. Határozzuk meg ennek a relációnak a tranzitív lezártját, azaz egy olyan Eljut(honnan, hova) relációt szeretnénk, amelyben két város akkor szerepel együtt, ha el lehet az egyikből a másikba jutni valahány átszállással.

Rekurzió

SQL3-as dolog, ideiglenes elképzelés

Lekérdezés és nem DDL (csak úgy kerül ide, hogy ez is SQL3)

Példa: Van egy Járat(honnan, hova) táblánk, amiben azt tároljuk, hogy mely városokból hova mennek közvetlenül gépek. Határozzuk meg ennek a relációnak a tranzitív lezártját, azaz egy olyan Eljut(honnan, hova) relációt szeretnénk, amelyben két város akkor szerepel együtt, ha el lehet az egyikből a másikba jutni valahány átszállással.

Ez relációs algebrában nem kifejezhető, de SQL3-ban igen.

Rekurzió

SQL3-as dolog, ideiglenes elképzelés

Lekérdezés és nem DDL (csak úgy kerül ide, hogy ez is SQL3)

Példa: Van egy Járat(honnan, hova) táblánk, amiben azt tároljuk, hogy mely városokból hova mennek közvetlenül gépek. Határozzuk meg ennek a relációnak a tranzitív lezártját, azaz egy olyan Eljut(honnan, hova) relációt szeretnénk, amelyben két város akkor szerepel együtt, ha el lehet az egyikből a másikba jutni valahány átszállással.

Ez relációs algebrában nem kifejezhető, de SQL3-ban igen.

```
WITH RECURSIVE Eljut(honnan, hova) AS
        (SELECT honnan, hova FROM Járat)
        UNION
        (SELECT Eljut AS R1, Eljut AS R2
                WHERE R1.hova = R2.honnan)
SELECT * FROM Eljut
```

Rekurzió

SQL3-as dolog, ideiglenes elképzelés

Lekérdezés és nem DDL (csak úgy kerül ide, hogy ez is SQL3)

Példa: Van egy Járat(honnan, hova) táblánk, amiben azt tároljuk, hogy mely városokból hova mennek közvetlenül gépek. Határozzuk meg ennek a relációnak a tranzitív lezártját, azaz egy olyan Eljut(honnan, hova) relációt szeretnénk, amelyben két város akkor szerepel együtt, ha el lehet az egyikből a másikba jutni valahány átszállással.

Ez relációs algebrában nem kifejezhető, de SQL3-ban igen.

```
WITH RECURSIVE Eljut(honnan, hova) AS
        (SELECT honnan, hova FROM Járat)
        UNION
        (SELECT Eljut AS R1, Eljut AS R2
                WHERE R1.hova = R2.honnan)
SELECT * FROM Eljut
```

Nem lehet bármi a rekurzív definícióban, pl. negációval óvatosan $\Longrightarrow$ nem biztonságos kifejezés

Adatbázisok elmélete

Funkcionális függőség

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

12. előadás

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)
Kérdés:

- Mik a jó relációk?

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)
Kérdés:

- Mik a jó relációk?
- Milyen relációkat érdemes tárolni?

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)
Kérdés:

- Mik a jó relációk?
- Milyen relációkat érdemes tárolni?
- Hogyan alakíthatunk tetszőleges relációkat jókká?

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)
Kérdés:

- Mik a jó relációk?
- Milyen relációkat érdemes tárolni?
- Hogyan alakíthatunk tetszőleges relációkat jókká?

Cél: El akarunk kerülni kellemetlen jelenségeket, **anomáliákat:**

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)
Kérdés:

- Mik a jó relációk?
- Milyen relációkat érdemes tárolni?
- Hogyan alakíthatunk tetszőleges relációkat jókká?

Cél: El akarunk kerülni kellemetlen jelenségeket, **anomáliákat:**

- *Módosítási anomália:* pl. ha a Termék(Termelő, Cím, Terméknév, Ár) reláció esetén egy termelő címe több sorban is előfordul, változáskor mindenhol át kell írni. Hiba esetén inkonzisztencia.

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)
Kérdés:

- Mik a jó relációk?
- Milyen relációkat érdemes tárolni?
- Hogyan alakíthatunk tetszőleges relációkat jókká?

Cél: El akarunk kerülni kellemetlen jelenségeket, **anomáliákat:**

- *Módosítási anomália:* pl. ha a Termék(Termelő, Cím, Terméknév, Ár) reláció esetén egy termelő címe több sorban is előfordul, változáskor mindenhol át kell írni. Hiba esetén inkonzisztencia.
- *Beszúrási anomália:* Nem tudunk beszúrni adatot, ha az egyik attribútum hiányzik, mert nem ismerjük (és nem lehet NULL).

Relációs sémák tervezése

Van elméleti alap $\Longrightarrow$ érv a relációs technika mellett
(Objektumosnak nincs ilyen.)
Kérdés:

- Mik a jó relációk?
- Milyen relációkat érdemes tárolni?
- Hogyan alakíthatunk tetszőleges relációkat jókká?

Cél: El akarunk kerülni kellemetlen jelenségeket, **anomáliákat:**

- *Módosítási anomália:* pl. ha a Termék(Termelő, Cím, Terméknév, Ár) reláció esetén egy termelő címe több sorban is előfordul, változáskor mindenhol át kell írni. Hiba esetén inkonzisztencia.
- *Beszúrási anomália:* Nem tudunk beszúrni adatot, ha az egyik attribútum hiányzik, mert nem ismerjük (és nem lehet NULL).
- *Törlési anomália:* Csak egész sorok törölhetők, így elveszhetnek hasznos adatok. Pl. ha egy termelő épp nem termel semmit, kitöröljük a címét is.

Relációs sémák tervezése

A relációk, tárolás jósága attól függ, hogy milyen megkötések vannak az adatokon.

Relációs sémák tervezése

A relációk, tárolás jósága attól függ, hogy milyen megkötések vannak az adatokon.
Megszorítások két osztálya:

- *Értékfüggő:* Pl. ÁR$\geq$ 0, ÉLETKOR egész $\leq$ 1000, NÉV karaktersor, CÍM$\neq$NULL, (típusleírások)

Relációs sémák tervezése

A relációk, tárolás jósága attól függ, hogy milyen megkötések vannak az adatokon.
Megszorítások két osztálya:

- *Értékfüggő:* Pl. ÁR$\geq$ 0, ÉLETKOR egész $\leq$ 1000, NÉV karaktersor, CÍM$\neq$NULL, (típusleírások)
- *Értékfüggetlen:* TERMÉKNÉV, TERMELŐ kulcs; $\forall$ TERMELŐ-nek egy címe van, egy TERMELŐ azonos nevű termékéből csak egy árú van

Relációs sémák tervezése

A relációk, tárolás jósága attól függ, hogy milyen megkötések vannak az adatokon.
Megszorítások két osztálya:

- *Értékfüggő:* Pl. ÁR$\geq$ 0, ÉLETKOR egész $\leq$ 1000, NÉV karaktersor, CÍM$\neq$NULL, (típusleírások)
- *Értékfüggetlen:* TERMÉKNÉV, TERMELŐ kulcs; $\forall$ TERMELŐ-nek egy címe van, egy TERMELŐ azonos nevű termékéből csak egy árú van

Utóbbi: az attribútumok mennyire függenek egymástól $\Longrightarrow$ **funkcionális függőség**

Funkcionális függőségek

Jelölés: $R(A_1, \ldots, A_n)$ reláció, X attribútum halmaz $\Longrightarrow X \subseteq R$
$X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ helyett $X = A_{i_1} A_{i_2} \ldots A_{i_k}$

Funkcionális függőségek

Jelölés: $R(A_1, \ldots, A_n)$ reláció, X attribútum halmaz $\Longrightarrow X \subseteq R$
$X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ helyett $X = A_{i_1} A_{i_2} \ldots A_{i_k}$

Definíció

*$Y \subseteq R$ **funkcionálisan függ** $X \subseteq R$-**től**, (jelölés: $X \to Y$), ha R bármely két sorára igaz, hogy ha ők megegyeznek X-en, akkor Y-on is megegyeznek.*

Funkcionális függőségek

Jelölés: $R(A_1, \ldots, A_n)$ reláció, X attribútum halmaz $\Longrightarrow X \subseteq R$
$X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ helyett $X = A_{i_1} A_{i_2} \ldots A_{i_k}$

Definíció

*$Y \subseteq R$ **funkcionálisan függ** $X \subseteq R$-**től**, (jelölés: $X \to Y$), ha R bármely két sorára igaz, hogy ha ők megegyeznek X-en, akkor Y-on is megegyeznek.*

Pl. X =TERMELŐ, TERMÉKNÉV; Y=ÁR $\Longrightarrow X \to Y$

Funkcionális függőségek

Jelölés: $R(A_1, \ldots, A_n)$ reláció, X attribútum halmaz $\Longrightarrow X \subseteq R$
$X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ helyett $X = A_{i_1} A_{i_2} \ldots A_{i_k}$

Definíció

*$Y \subseteq R$ **funkcionálisan függ** $X \subseteq R$-**től**, (jelölés: $X \to Y$), ha R bármely két sorára igaz, hogy ha ők megegyeznek X-en, akkor Y-on is megegyeznek.*

Pl. X =TERMELŐ, TERMÉKNÉV; Y=ÁR $\Longrightarrow X \to Y$

Megjegyzések:

- Azok az érdekes összefüggések, amik minden ilyen attribútumokkal rendelkező táblában fenn kell, hogy álljanak: axiómaszerű feltételek, az adatbázis bármely változása esetén is fennállnak $\Longrightarrow$ *érdemi függés*

Funkcionális függőségek

Jelölés: $R(A_1, \ldots, A_n)$ reláció, X attribútum halmaz $\Longrightarrow X \subseteq R$
$X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ helyett $X = A_{i_1} A_{i_2} \ldots A_{i_k}$

Definíció

*$Y \subseteq R$ **funkcionálisan függ** $X \subseteq R$-**től**, (jelölés: $X \to Y$), ha R bármely két sorára igaz, hogy ha ők megegyeznek X-en, akkor Y-on is megegyeznek.*

Pl. X =TERMELŐ, TERMÉKNÉV; Y=ÁR $\Longrightarrow X \to Y$
Megjegyzések:

- Azok az érdekes összefüggések, amik minden ilyen attribútumokkal rendelkező táblában fenn kell, hogy álljanak: axiómaszerű feltételek, az adatbázis bármely változása esetén is fennállnak $\Longrightarrow$ *érdemi függés*
  Azok, amik csak véletlenül, csak egy pillanatban állnak fenn $\Longrightarrow$ *eseti függés*
  (ezek nem érdekelnek, például lehetséges hogy egy adott pillanatban minden ár csak egyszer szerepel és ekkor úgy tűnik, mintha Ár$\to$ Termék érvényes függés lenne)

Funkcionális függőségek

Jelölés: $R(A_1, \ldots, A_n)$ reláció, X attribútum halmaz $\Longrightarrow X \subseteq R$
$X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ helyett $X = A_{i_1} A_{i_2} \ldots A_{i_k}$

Definíció

*$Y \subseteq R$ **funkcionálisan függ** $X \subseteq R$-**től**, (jelölés: $X \to Y$), ha R bármely két sorára igaz, hogy ha ők megegyeznek X-en, akkor Y-on is megegyeznek.*

Pl. X =TERMELŐ, TERMÉKNÉV; Y=ÁR $\Longrightarrow X \to Y$
Megjegyzések:

- Azok az érdekes összefüggések, amik minden ilyen attribútumokkal rendelkező táblában fenn kell, hogy álljanak: axiómaszerű feltételek, az adatbázis bármely változása esetén is fennállnak $\Longrightarrow$ *érdemi függés*
  Azok, amik csak véletlenül, csak egy pillanatban állnak fenn $\Longrightarrow$ *eseti függés*
  (ezek nem érdekelnek, például lehetséges hogy egy adott pillanatban minden ár csak egyszer szerepel és ekkor úgy tűnik, mintha Ár$\to$ Termék érvényes függés lenne)
- Tehát az érdemi függések megadása modellezési kérdés: a séma megadásakor döntjük el, hogy milyen függéseket akarunk fenntartani mindenáron.

Funkcionális függőségek

Jelölés: $R(A_1, \ldots, A_n)$ reláció, X attribútum halmaz $\Longrightarrow X \subseteq R$
$X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ helyett $X = A_{i_1} A_{i_2} \ldots A_{i_k}$

Definíció

*$Y \subseteq R$ **funkcionálisan függ** $X \subseteq R$**-től**, (jelölés: $X \to Y$), ha R bármely két sorára igaz, hogy ha ők megegyeznek X-en, akkor Y-on is megegyeznek.*

Pl. X =TERMELŐ, TERMÉKNÉV; Y=ÁR $\Longrightarrow X \to Y$
Megjegyzések:

- Azok az érdekes összefüggések, amik minden ilyen attribútumokkal rendelkező táblában fenn kell, hogy álljanak: axiómaszerű feltételek, az adatbázis bármely változása esetén is fennállnak $\Longrightarrow$ *érdemi függés*
  Azok, amik csak véletlenül, csak egy pillanatban állnak fenn $\Longrightarrow$ *eseti függés*
  (ezek nem érdekelnek, például lehetséges hogy egy adott pillanatban minden ár csak egyszer szerepel és ekkor úgy tűnik, mintha Ár$\to$ Termék érvényes függés lenne)
- Tehát az érdemi függések megadása modellezési kérdés: a séma megadásakor döntjük el, hogy milyen függéseket akarunk fenntartani mindenáron.
  Ezentúl a relációs sémának része lesz a függőségek halmaza F is $\Longrightarrow (R, F)$
  Vagyis megadjuk, hogy mik a séma attribútumai és mik az érdemi függései.

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

S(NÉV, CÍM, VÁROS, IRÁNYÍTÓSZ, TELEFON)

CÍM, VÁROS $\rightarrow$ IRÁNYÍTÓSZ
IRÁNYÍTÓSZ $\rightarrow$ VÁROS
NÉV, CÍM, VÁROS $\rightarrow$ TELEFON

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

S(NÉV, CÍM, VÁROS, IRÁNYÍTÓSZ, TELEFON)

CÍM, VÁROS $\rightarrow$ IRÁNYÍTÓSZ
IRÁNYÍTÓSZ $\rightarrow$ VÁROS
NÉV, CÍM, VÁROS $\rightarrow$ TELEFON

- Egy adott reláció adott állapotából nem következik semmilyen érdemi függés.

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

S(NÉV, CÍM, VÁROS, IRÁNYÍTÓSZ, TELEFON)

CÍM, VÁROS $\rightarrow$ IRÁNYÍTÓSZ
IRÁNYÍTÓSZ $\rightarrow$ VÁROS
NÉV, CÍM, VÁROS $\rightarrow$ TELEFON

- Egy adott reláció adott állapotából nem következik semmilyen érdemi függés. Viszont látszódhat olyan, hogy mi nem függhet mitől.

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

S(NÉV, CÍM, VÁROS, IRÁNYÍTÓSZ, TELEFON)

CÍM, VÁROS $\rightarrow$ IRÁNYÍTÓSZ
IRÁNYÍTÓSZ $\rightarrow$ VÁROS
NÉV, CÍM, VÁROS $\rightarrow$ TELEFON

- Egy adott reláció adott állapotából nem következik semmilyen érdemi függés. Viszont látszódhat olyan, hogy mi nem függhet mitől.
- $X \rightarrow Y$ teljesülhet úgy is, hogy az adott relációban nincs is két olyan sor, amik X-en megegyeznek.

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

S(NÉV, CÍM, VÁROS, IRÁNYÍTÓSZ, TELEFON)

CÍM, VÁROS $\rightarrow$ IRÁNYÍTÓSZ
IRÁNYÍTÓSZ $\rightarrow$ VÁROS
NÉV, CÍM, VÁROS $\rightarrow$ TELEFON

- Egy adott reláció adott állapotából nem következik semmilyen érdemi függés. Viszont látszódhat olyan, hogy mi nem függhet mitől.
- $X \rightarrow Y$ teljesülhet úgy is, hogy az adott relációban nincs is két olyan sor, amik X-en megegyeznek.
- X-nek és Y-nak nem kell diszjunktaknak lenniük

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

S(NÉV, CÍM, VÁROS, IRÁNYÍTÓSZ, TELEFON)

CÍM, VÁROS $\rightarrow$ IRÁNYÍTÓSZ
IRÁNYÍTÓSZ $\rightarrow$ VÁROS
NÉV, CÍM, VÁROS $\rightarrow$ TELEFON

- Egy adott reláció adott állapotából nem következik semmilyen érdemi függés. Viszont látszódhat olyan, hogy mi nem függhet mitől.
- $X \rightarrow Y$ teljesülhet úgy is, hogy az adott relációban nincs is két olyan sor, amik X-en megegyeznek.
- X-nek és Y-nak nem kell diszjunktaknak lenniük

A séma megadása csak a keretet jelenti, beleértve a függeseket is, ha ezt feltöltjük adatokkal, akkor kapunk egy a sémára illeszkedő relációt.

Funkcionális függőségek

R(TERMELŐ, TERMÉKNÉV, ÁR, CÍM)

TERMELŐ, TERMÉKNÉV $\rightarrow$ TERMELŐ, TERMÉKNÉV, ÁR, CÍM
TERMELŐ $\rightarrow$ CÍM

S(NÉV, CÍM, VÁROS, IRÁNYÍTÓSZ, TELEFON)

CÍM, VÁROS $\rightarrow$ IRÁNYÍTÓSZ
IRÁNYÍTÓSZ $\rightarrow$ VÁROS
NÉV, CÍM, VÁROS $\rightarrow$ TELEFON

- Egy adott reláció adott állapotából nem következik semmilyen érdemi függés. Viszont látszódhat olyan, hogy mi nem függhet mitől.
- $X \rightarrow Y$ teljesülhet úgy is, hogy az adott relációban nincs is két olyan sor, amik X-en megegyeznek.
- X-nek és Y-nak nem kell diszjunktaknak lenniük

A séma megadása csak a keretet jelenti, beleértve a függeseket is, ha ezt feltöltjük adatokkal, akkor kapunk egy a sémára illeszkedő relációt. A r reláció akkor illeszkedik az (R, F) sémára ha az attribútumai az R-ben adottak és teljesülnek benne az F függések.

Logikai következmény

Kérdés: ha adott egy F függéshalmaz és egy reláció, amiben F függései igazak, akkor milyen további függések lesznek még biztosan igazak?

Logikai következmény

Kérdés: ha adott egy F függéshalmaz és egy reláció, amiben F függései igazak, akkor milyen további függések lesznek még biztosan igazak?
Például: ha HALLGATÓ, TÁRGY $\rightarrow$ GYAKORLAT és GYAKORLAT $\rightarrow$ GYAKVEZ, akkor HALLGATÓ, TÁRGY $\rightarrow$ GYAKVEZ.

Logikai következmény

Kérdés: ha adott egy F függéshalmaz és egy reláció, amiben F függései igazak, akkor milyen további függések lesznek még biztosan igazak?
Például: ha HALLGATÓ, TÁRGY $\rightarrow$ GYAKORLAT és GYAKORLAT $\rightarrow$ GYAKVEZ, akkor HALLGATÓ, TÁRGY $\rightarrow$ GYAKVEZ.
Azaz általánosabban: ha $XY \rightarrow Z$ és $Z \rightarrow W$, akkor attól függetlenül, hogy mi a reláció és X, Y, Z, W, igaz lesz, hogy $XY \rightarrow W$.

Logikai következmény

Kérdés: ha adott egy F függéshalmaz és egy reláció, amiben F függései igazak, akkor milyen további függések lesznek még biztosan igazak?
Például: ha HALLGATÓ, TÁRGY $\rightarrow$ GYAKORLAT és GYAKORLAT $\rightarrow$ GYAKVEZ, akkor HALLGATÓ, TÁRGY $\rightarrow$ GYAKVEZ.
Azaz általánosabban: ha $XY \rightarrow Z$ és $Z \rightarrow W$, akkor attól függetlenül, hogy mi a reláció és X, Y, Z, W, igaz lesz, hogy $XY \rightarrow W$.

Definíció

*Adott (R, F). Az $X \rightarrow Y$ függés **logikai következménye** (szemantikai következménye) F-nek, ha az $X \rightarrow Y$ minden olyan r relációban teljesül, ahol F függései mind teljesülnek.*
Jelölése: $F \models X \rightarrow Y$

Logikai következmény

Kérdés: ha adott egy F függéshalmaz és egy reláció, amiben F függései igazak, akkor milyen további függések lesznek még biztosan igazak?
Például: ha HALLGATÓ, TÁRGY $\rightarrow$ GYAKORLAT és GYAKORLAT $\rightarrow$ GYAKVEZ, akkor HALLGATÓ, TÁRGY $\rightarrow$ GYAKVEZ.
Azaz általánosabban: ha $XY \rightarrow Z$ és $Z \rightarrow W$, akkor attól függetlenül, hogy mi a reláció és X, Y, Z, W, igaz lesz, hogy $XY \rightarrow W$.

Definíció

*Adott (R, F). Az $X \rightarrow Y$ függés **logikai következménye** (szemantikai következménye) F-nek, ha az $X \rightarrow Y$ minden olyan r relációban teljesül, ahol F függései mind teljesülnek.*
Jelölése: $F \models X \rightarrow Y$

Azaz ez a fogalom azt adja meg, hogy mely függéseknek kell szükségszerűen teljesülniük minden olyan sémában/relációban, ahol F függései fennállnak.

Logikai következmény

Kérdés: ha adott egy F függéshalmaz és egy reláció, amiben F függései igazak, akkor milyen további függések lesznek még biztosan igazak?
Például: ha HALLGATÓ, TÁRGY $\rightarrow$ GYAKORLAT és GYAKORLAT $\rightarrow$ GYAKVEZ, akkor HALLGATÓ, TÁRGY $\rightarrow$ GYAKVEZ.
Azaz általánosabban: ha $XY \rightarrow Z$ és $Z \rightarrow W$, akkor attól függetlenül, hogy mi a reláció és X, Y, Z, W, igaz lesz, hogy $XY \rightarrow W$.

Definíció

*Adott (R, F). Az $X \rightarrow Y$ függés **logikai következménye** (szemantikai következménye) F-nek, ha az $X \rightarrow Y$ minden olyan r relációban teljesül, ahol F függései mind teljesülnek.*
Jelölése: $F \models X \rightarrow Y$

Azaz ez a fogalom azt adja meg, hogy mely függéseknek kell szükségszerűen teljesülniük minden olyan sémában/relációban, ahol F függései fennállnak.

Hogyan lehetne ezeket meghatározni, illetve eldönteni, hogy egy függés ilyen-e?

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.
Levezethetőség jele: $F \vdash X \to Y$

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.

Levezethetőség jele: $F \vdash X \to Y$

Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.
Levezethetőség jele: $F \vdash X \to Y$
Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.
(Pl. logikában így van.)

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.
Levezethetőség jele: $F \vdash X \to Y$
Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.
(Pl. logikában így van.)
$\models \Rightarrow \vdash$: *Teljességi tétel*, ami igaz az levezethető.

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.

Levezethetőség jele: $F \vdash X \to Y$

Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.

(Pl. logikában így van.)

$\models \Rightarrow \vdash$: *Teljességi tétel*, ami igaz az levezethető.

$\vdash \Rightarrow \models$: *Igazság tétel*, csak igaz dolgok vezethetők le.

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.
Levezethetőség jele: $F \vdash X \to Y$
Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.
(Pl. logikában így van.)
$\models \Rightarrow \vdash$: *Teljességi tétel*, ami igaz az levezethető.
$\vdash \Rightarrow \models$: *Igazság tétel*, csak igaz dolgok vezethetők le.

Definíció

Egy $X \to Y$ függőség akkor vezethető le egy adott F függőséghalmazból, ha az axiómák ismételt alkalmazásával F-ből megkapjuk $X \to Y$-t. Jele: $F \vdash X \to Y$.

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.
Levezethetőség jele: $F \vdash X \to Y$
Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.
(Pl. logikában így van.)
$\models \Rightarrow \vdash$: *Teljességi tétel*, ami igaz az levezethető.
$\vdash \Rightarrow \models$: *Igazság tétel*, csak igaz dolgok vezethetők le.

Definíció

Egy $X \to Y$ függőség akkor vezethető le egy adott F függőséghalmazból, ha az axiómák ismételt alkalmazásával F-ből megkapjuk $X \to Y$-t. Jele: $F \vdash X \to Y$.

Armstrong-axiómák

1. Reflexivitás: Ha $X, Y \subseteq R$ és $Y \subseteq X$, akkor $X \to Y$.

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.
Levezethetőség jele: $F \vdash X \to Y$
Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.
(Pl. logikában így van.)
$\models \Rightarrow \vdash$: *Teljességi tétel*, ami igaz az levezethető.
$\vdash \Rightarrow \models$: *Igazság tétel*, csak igaz dolgok vezethetők le.

Definíció

Egy $X \to Y$ függőség akkor vezethető le egy adott F függőséghalmazból, ha az axiómák ismételt alkalmazásával F-ből megkapjuk $X \to Y$-t. Jele: $F \vdash X \to Y$.

Armstrong-axiómák

1. Reflexivitás: Ha $X, Y \subseteq R$ és $Y \subseteq X$, akkor $X \to Y$.
2. Kiegészítési tulajdonság: Ha $X, Y \subseteq R$ és $X \to Y$, akkor $XW \to YW$ igaz tetszőleges $W \subseteq R$-re.

Logikai következmény

Felveszünk axiómákat, és azok segítségével próbáljuk levezetni. Persze ehhez az kell, hogy pontosan azokat lehessen levezetni F-ből, amik logikai következményei neki.
Levezethetőség jele: $F \vdash X \to Y$
Tehát bevezetünk axiómákat, levezethetőséget és belátjuk, hogy $\models \iff \vdash$.
(Pl. logikában így van.)
$\models \Rightarrow \vdash$: *Teljességi tétel*, ami igaz az levezethető.
$\vdash \Rightarrow \models$: *Igazság tétel*, csak igaz dolgok vezethetők le.

Definíció

Egy $X \to Y$ függőség akkor vezethető le egy adott F függőséghalmazból, ha az axiómák ismételt alkalmazásával F-ből megkapjuk $X \to Y$-t. Jele: $F \vdash X \to Y$.

Armstrong-axiómák

1. Reflexivitás: Ha $X, Y \subseteq R$ és $Y \subseteq X$, akkor $X \to Y$.
2. Kiegészítési tulajdonság: Ha $X, Y \subseteq R$ és $X \to Y$, akkor $XW \to YW$ igaz tetszőleges $W \subseteq R$-re.
3. Tranzitivitás: Ha $X, Y, Z \subseteq R$, $X \to Y$ és $Y \to Z$, akkor $X \to Z$.

Igazságtétel bizonyítása

Bizonyítás.

(Igazság tétel)

Azt kell belátni, hogy ha egy függés (esetleg több lépésben) levezethető F-ből a három axióma segítségével, akkor ez a függés logikai következménye is F-nek,

Igazságtétel bizonyítása

Bizonyítás.

(Igazság tétel)

Azt kell belátni, hogy ha egy függés (esetleg több lépésben) levezethető F-ből a három axióma segítségével, akkor ez a függés logikai következménye is F-nek, azaz minden olyan relációban, ahol F minden függése teljesül, ott teljesül a levezetett függés is.

Igazságtétel bizonyítása

Bizonyítás.

(Igazság tétel)

Azt kell belátni, hogy ha egy függés (esetleg több lépésben) levezethető F-ből a három axióma segítségével, akkor ez a függés logikai következménye is F-nek, azaz minden olyan relációban, ahol F minden függése teljesül, ott teljesül a levezetett függés is. Ehhez elég azt belátni, hogy külön-külön, az egyes axiómák egyszeri használata ilyen függést ad.

Igazságtétel bizonyítása

Bizonyítás.

(Igazság tétel)

Azt kell belátni, hogy ha egy függés (esetleg több lépésben) levezethető F-ből a három axióma segítségével, akkor ez a függés logikai következménye is F-nek, azaz minden olyan relációban, ahol F minden függése teljesül, ott teljesül a levezetett függés is. Ehhez elég azt belátni, hogy külön-külön, az egyes axiómák egyszeri használata ilyen függést ad.

1. Reflexivitás: Azt kell belátni, hogy minden r relációban, minden $Y \subseteq X \subseteq R$ attribútumhalmaz esetén $X \to Y$ igaz, azaz ha r bármely két adott sora megegyezik X-en, akkor megegyeznek Y-on is. De mivel $Y \subseteq X$, ezért nyilván megegyeznek Y-on, ha X-en megegyeztek. $\surd$

Igazságtétel bizonyítása

Bizonyítás.

(Igazság tétel)

Azt kell belátni, hogy ha egy függés (esetleg több lépésben) levezethető F-ből a három axióma segítségével, akkor ez a függés logikai következménye is F-nek, azaz minden olyan relációban, ahol F minden függése teljesül, ott teljesül a levezetett függés is. Ehhez elég azt belátni, hogy külön-külön, az egyes axiómák egyszeri használata ilyen függést ad.

1. Reflexivitás: Azt kell belátni, hogy minden r relációban, minden $Y \subseteq X \subseteq R$ attribútumhalmaz esetén $X \to Y$ igaz, azaz ha r bármely két adott sora megegyezik X-en, akkor megegyeznek Y-on is. De mivel $Y \subseteq X$, ezért nyilván megegyeznek Y-on, ha X-en megegyeztek. $\surd$
2. Kiegészítési tulajdonság: Az kell, hogy ha egy R-re illeszkedő r relációban $X \to Y$ igaz, akkor $XW \to YW$ is igaz lesz. Vegyünk két sort r-ben, ami megegyezik XW-n. Ekkor ezek megegyeznek X-en és W-n is, külön-külön. Mivel $X \to Y$, így megegyeznek Y-n is, tehát YW-n is. $\surd$

Igazságtétel bizonyítása

Bizonyítás.

(Igazság tétel)

Azt kell belátni, hogy ha egy függés (esetleg több lépésben) levezethető F-ből a három axióma segítségével, akkor ez a függés logikai következménye is F-nek, azaz minden olyan relációban, ahol F minden függése teljesül, ott teljesül a levezetett függés is. Ehhez elég azt belátni, hogy külön-külön, az egyes axiómák egyszeri használata ilyen függést ad.

1. Reflexivitás: Azt kell belátni, hogy minden r relációban, minden $Y \subseteq X \subseteq R$ attribútumhalmaz esetén $X \to Y$ igaz, azaz ha r bármely két adott sora megegyezik X-en, akkor megegyeznek Y-on is. De mivel $Y \subseteq X$, ezért nyilván megegyeznek Y-on, ha X-en megegyeztek. $\surd$
2. Kiegészítési tulajdonság: Az kell, hogy ha egy R-re illeszkedő r relációban $X \to Y$ igaz, akkor $XW \to YW$ is igaz lesz. Vegyünk két sort r-ben, ami megegyezik XW-n. Ekkor ezek megegyeznek X-en és W-n is, külön-külön. Mivel $X \to Y$, így megegyeznek Y-n is, tehát YW-n is. $\surd$
3. Tranzitivitás: Az kell, hogy ha egy R-re illeszkedő r relációban $X \to Y$ és $Y \to Z$ igaz, akkor $X \to Z$ is igaz lesz. Vegyünk két sort, ami megegyezik X-en. Mivel $X \to Y$, megegyeznek Y-n is. De mivel $Y \to Z$, megegyeznek Z-n is. $\surd$

$\square$

Példa

Állítás

Ha $R(\mathit{Város}, \mathit{Utca}, \mathit{Irányítószám})$ *és* $F = \{VU \to I, I \to V\}$, *akkor* $F \vdash IU \to VIU$ *(és mivel* $\vdash \Rightarrow \models$*-t már láttuk, ezért* $F \models IU \to VIU$*).*

$\surd$

Példa

Állítás

Ha $R(\mathit{Város}, \mathit{Utca}, \mathit{Irányítószám})$ *és* $F = \{\mathit{VU} \to \mathit{I}, \mathit{I} \to \mathit{V}\}$, *akkor* $F \vdash \mathit{IU} \to \mathit{VIU}$ *(és mivel* $\vdash \Rightarrow \models$*-t már láttuk, ezért* $F \models \mathit{IU} \to \mathit{VIU}$*).*

$\surd$

Bizonyítás.

i) $I \to V$: ez F-beli

ii) $IU \to VU$: kiegészítve U-val

iii) $IU \to IVU$: kiegészítve I-vel

□

Levezethető szabályok

Néhány további szabály, ami levezethető az axiómákból (és az igazságtétel miatt igazak is).

Levezethető szabályok

Néhány további szabály, ami levezethető az axiómákból (és az igazságtétel miatt igazak is).

Állítás (Unió szabály)

$\{X \to Y,\ X \to Z\} \vdash X \to YZ$

Levezethető szabályok

Néhány további szabály, ami levezethető az axiómákból (és az igazságtétel miatt igazak is).

Állítás (Unió szabály)

$\{X \to Y,\ X \to Z\} \vdash X \to YZ$

Bizonyítás.

i) $X \to Y$: ez F-beli

ii) $XZ \to YZ$: kiegészítve Z-val

iii) $X \to Z$: ez F-beli

iv) $X \to XZ$: kiegészítve X-vel

v) $X \to YZ$: iv) és ii) + tranzitivitás

□

Levezethető szabályok

Állítás (Áltranzitiv szabály)

$\{X \to Y,\ YW \to Z\} \vdash XW \to Z$

Levezethető szabályok

Állítás (Általranzitiv szabály)

$\{X \to Y,\ YW \to Z\} \vdash XW \to Z$

Bizonyítás.

i) $X \to Y$: ez F-beli

ii) $XW \to YW$: kiegészítve W-val

iii) $YW \to Z$: ez F-beli

iv) $XW \to Z$: ii) és iii) + tranzitivitás

Levezethető szabályok

Állítás (Áltranzitiv szabály)

$\{X \to Y,\ YW \to Z\} \vdash XW \to Z$

Bizonyítás.

i) $X \to Y$: ez F-beli

ii) $XW \to YW$: kiegészítve W-val

iii) $YW \to Z$: ez F-beli

iv) $XW \to Z$: ii) és iii) + tranzitivitás

□

Levezethető szabályok

Állítás (Felbontási szabály)

Tegyük fel, hogy $Z \subseteq Y$, ekkor $\{X \to Y\} \vdash X \to Z$

Levezethető szabályok

Állítás (Felbontási szabály)

Tegyük fel, hogy $Z \subseteq Y$, ekkor $\{X \to Y\} \vdash X \to Z$

Bizonyítás.

i) $X \to Y$: ez F-beli

ii) $Y \to Z$: reflexivitás

iii) $X \to Z$: i) és ii) + tranzitivitás

□

Lezárás

Definíció

*Ha F egy függéshalamaz, akkor a **lezártja** F^+ az F-ből levezethető összes függés:*

$$F^+ = \{X \to Y \mid F \vdash X \to Y\}$$

Lezárás

Definíció

Ha F egy függéshalamaz, akkor a ***lezártja*** F^+ *az F-ből levezethető összes függés:*

$$F^+ = \{X \to Y \mid F \vdash X \to Y\}$$

Jó: mert ha majd belátjuk $\models \iff \vdash$-t, akkor kiderül, hogy ez éppen az F-ből szükségszerűen következő összes függést adja meg.

Lezárás

Definíció

Ha F egy függéshalamaz, akkor a ***lezártja*** F^+ *az F-ből levezethető összes függés:*

$$F^+ = \{X \to Y \mid F \vdash X \to Y\}$$

Jó: mert ha majd belátjuk $\models \iff \vdash$-t, akkor kiderül, hogy ez éppen az F-ből szükségszerűen következő összes függést adja meg.

Gond: nagyon nagy lehet
Pl. $R(A_1, \ldots, A_n, B_1, \ldots B_n)$ és $F = \{A_i \to B_j \mid 1 \leq i, j \leq n\}$, akkor ez n^2 db függés

Lezárás

Definíció

Ha F egy függéshalamaz, akkor a ***lezártja*** F^+ *az F-ből levezethető összes függés:*

$$F^+ = \{X \to Y \mid F \vdash X \to Y\}$$

Jó: mert ha majd belátjuk $\models \iff \vdash$-t, akkor kiderül, hogy ez éppen az F-ből szükségszerűen következő összes függést adja meg.

Gond: nagyon nagy lehet
Pl. $R(A_1, \ldots, A_n, B_1, \ldots B_n)$ és $F = \{A_i \to B_j \mid 1 \leq i, j \leq n\}$, akkor ez n^2 db függés F^+-ban benne van minden $A_{i_1} \ldots A_{i_k} \to B_{j_1} \ldots B_{j_l}$, azaz $(2^n - 1)(2^n - 1) \approx 2^{2n}$ eleme van.

Lezárás

Ezért ehelyett valami mást nézünk, amit könnyebb lesz meghatározni és jól közelíti F^+-t:

Lezárás

Ezért ehelyett valami mást nézünk, amit könnyebb lesz meghatározni és jól közelíti F^+-t:

Definíció

Ha X egy attribútum halmaz (R, F)-ben, akkor ***lezártja***

$$X^+(F) = \{A \in R \mid F \vdash X \to A\},$$

azaz azon attribútumok, amik függnek X-től.

Attribútumhalmaz lezárása

Állítás

$X \subseteq X^+(F) \subseteq R$

Attribútumhalmaz lezárása

Állítás

$X \subseteq X^+(F) \subseteq R$

Lemma

(Fontos!!!) $F \vdash X \to Y \iff Y \subseteq X^+(F)$

Attribútumhalmaz lezárása

Állítás

$X \subseteq X^+(F) \subseteq R$

Lemma

(Fontos!!!) $F \vdash X \to Y \iff Y \subseteq X^+(F)$

Bizonyítás.

$\boxed{\Longrightarrow:}$ Tegyük fel, hogy $F \vdash X \to Y$ és legyen $A \in Y$.
$F \vdash X \to A$, hiszen vegyük $X \to Y$ levezetését és alkalmazzuk a felbontási szabályt a végén.
Definíció szerint ekkor $A \in X^+(F)$. Ez minden $A \in Y$-ra igaz. $\surd$

Attribútumhalmaz lezárása

Állítás

$X \subseteq X^+(F) \subseteq R$

Lemma

(Fontos!!!) $F \vdash X \to Y \iff Y \subseteq X^+(F)$

Bizonyítás.

$\boxed{\Longrightarrow:}$ Tegyük fel, hogy $F \vdash X \to Y$ és legyen $A \in Y$.
$F \vdash X \to A$, hiszen vegyük $X \to Y$ levezetését és alkalmazzuk a felbontási szabályt a végén.
Definíció szerint ekkor $A \in X^+(F)$. Ez minden $A \in Y$-ra igaz. $\surd$
$\boxed{\Longleftarrow:}$ Legyen $Y = A_1 \ldots A_k \subseteq X^+(F)$.
Így definíció szerint $\forall A_i \in Y$-ra $F \vdash X \to A_i$.
Ekkor $X \to Y$ levezetése: vesszük az A_i-k levezetését és a végén alkalmazzuk az uniós szabályt $k-1$-szer. $\surd$

Attribútumhalmaz lezárása

Állítás

$X \subseteq X^+(F) \subseteq R$

Lemma

(Fontos!!!) $F \vdash X \to Y \iff Y \subseteq X^+(F)$

Bizonyítás.

$\boxed{\Longrightarrow:}$ Tegyük fel, hogy $F \vdash X \to Y$ és legyen $A \in Y$.
$F \vdash X \to A$, hiszen vegyük $X \to Y$ levezetését és alkalmazzuk a felbontási szabályt a végén.
Definíció szerint ekkor $A \in X^+(F)$. Ez minden $A \in Y$-ra igaz. $\surd$
$\boxed{\Longleftarrow:}$ Legyen $Y = A_1 \ldots A_k \subseteq X^+(F)$.
Így definíció szerint $\forall A_i \in Y$-ra $F \vdash X \to A_i$.
Ekkor $X \to Y$ levezetése: vesszük az A_i-k levezetését és a végén alkalmazzuk az uniós szabályt $k-1$-szer. $\surd$ $\square$

Attribútumhalmaz lezárása

Következménye: Ha minden X-re ismerjük/ki tudjuk számítani $X^+(F)$-et, akkor tetszőleges $X \to Y$ függésről eldönthető, hogy F^+-beli-e vagy sem,

Attribútumhalmaz lezárása

Következménye: Ha minden X-re ismerjük/ki tudjuk számítani $X^+(F)$-et, akkor tetszőleges $X \to Y$ függésről eldönthető, hogy F^+-beli-e vagy sem, mert $X \to Y \in F^+$ pontosan akkor teljesül (definíció szerint), ha $F \vdash X \to Y$, de ez meg az előbbi lemma szerint pontosan akkor van, ha $Y \subseteq X^+(F)$

Attribútumhalmaz lezárása

Következménye: Ha minden X-re ismerjük/ki tudjuk számítani $X^+(F)$-et, akkor tetszőleges $X \to Y$ függésről eldönthető, hogy F^+-beli-e vagy sem, mert $X \to Y \in F^+$ pontosan akkor teljesül (definíció szerint), ha $F \vdash X \to Y$, de ez meg az előbbi lemma szerint pontosan akkor van, ha $Y \subseteq X^+(F)$

Megjegyzés: Majd látjuk, hogy $X^+(F)$ kiszámolására lesz gyors algoritmus.

Teljességi tétel

Tétel (Teljességi tétel)

Ha $F \models X \to Y$, akkor $F \vdash X \to Y$.

Teljességi tétel

Tétel (Teljességi tétel)

Ha $F \models X \to Y$, akkor $F \vdash X \to Y$.

Bizonyítás.

Tegyük fel indirekt, hogy van olyan $X \to Y$ függés és F függéshalmaz, hogy $X \to Y$ nem vezethető le F-ből ($F \not\vdash X \to Y$), noha logikai következménye neki ($F \models X \to Y$).

Teljességi tétel

Tétel (Teljességi tétel)

Ha $F \models X \to Y$, akkor $F \vdash X \to Y$.

Bizonyítás.

Tegyük fel indirekt, hogy van olyan $X \to Y$ függés és F függéshalmaz, hogy $X \to Y$ nem vezethető le F-ből ($F \not\vdash X \to Y$), noha logikai következménye neki ($F \models X \to Y$). Ez utóbbi azt jelenti, hogy minden olyan relációban, amiben F függőségei teljesülnek, ha X-en megegyezik két sor, akkor azok megegyeznek Y-on is.

Teljességi tétel

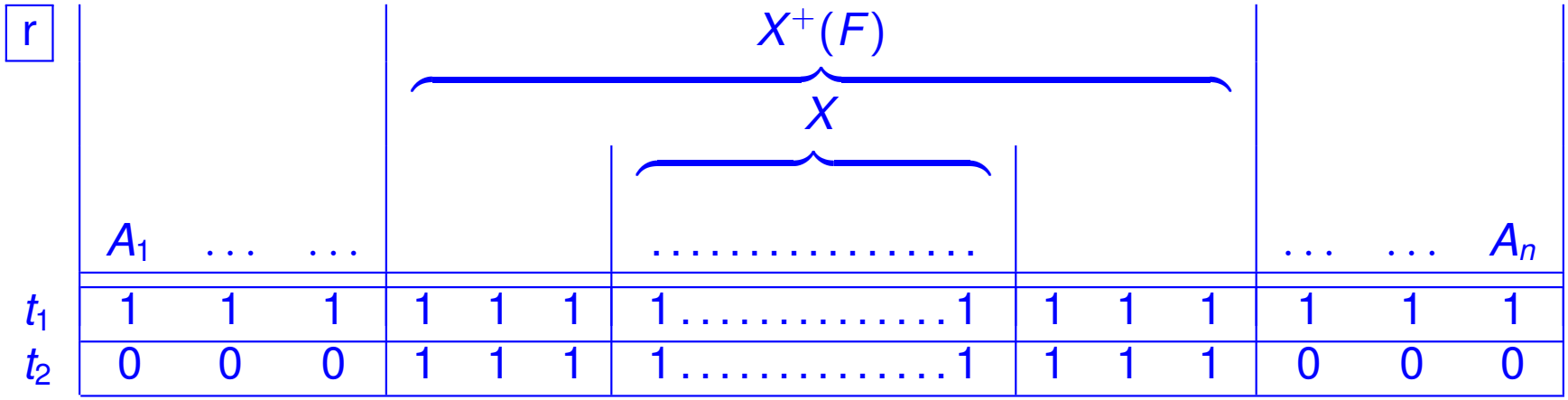
Tétel (Teljességi tétel)

Ha $F \models X \to Y$, akkor $F \vdash X \to Y$.

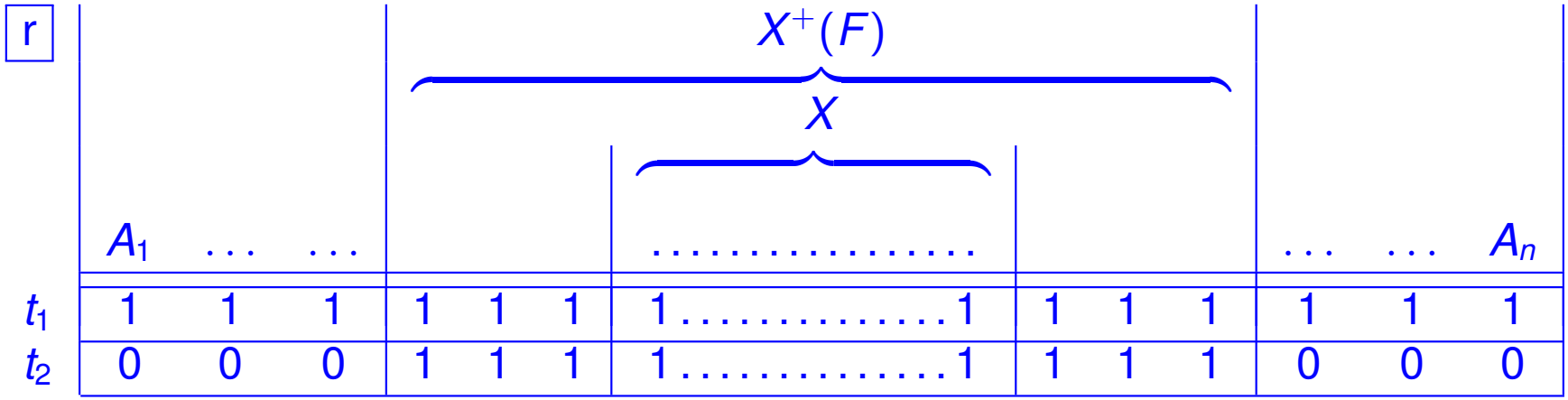
Bizonyítás.

Tegyük fel indirekt, hogy van olyan $X \to Y$ függés és F függéshalmaz, hogy $X \to Y$ nem vezethető le F-ből ($F \not\vdash X \to Y$), noha logikai következménye neki ($F \models X \to Y$). Ez utóbbi azt jelenti, hogy minden olyan relációban, amiben F függőségei teljesülnek, ha X-en megegyezik két sor, akkor azok megegyeznek Y-on is.
Úgy jutunk ellentmondásra, hogy konstruálunk egy olyan r relációt, ahol F függőségei teljesülnek, de $X \not\to Y$, ami ellentmond $F \models X \to Y$-nak.

Bizonyítás (folyt.)

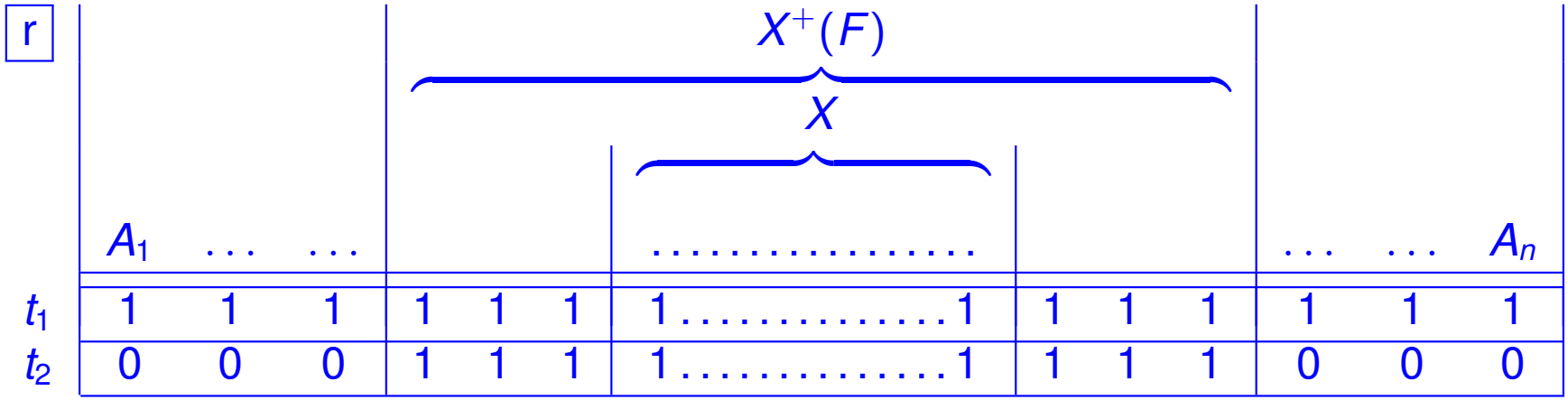


Bizonyítás (folyt.)



Tehát r két soros reláció, $X^+(F)$-en megegyezik a két sor, a többin különbözik.

Bizonyítás (folyt.)

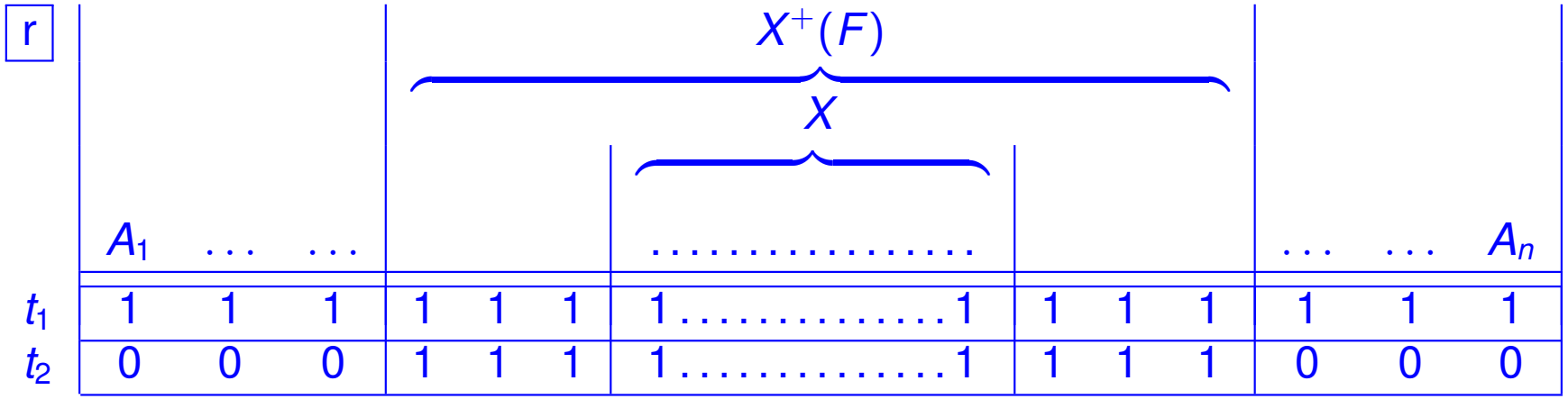


Tehát r két soros reláció, $X^+(F)$-en megegyezik a két sor, a többin különbözik.

Állítás

r-ben teljesülnek F függései.

Bizonyítás (folyt.)



Tehát r két soros reláció, $X^+(F)$-en megegyezik a két sor, a többin különbözik.

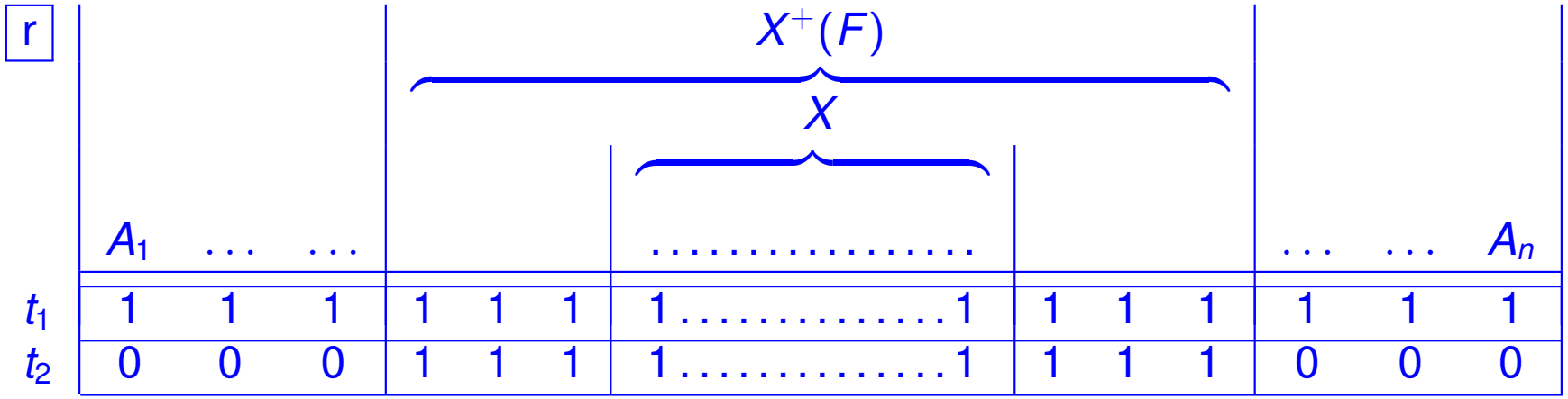
Állítás

r-ben teljesülnek F függései.

Bizonyítás.

Legyen $U \to V \in F$.
Ha $U \not\subseteq X^+(F) \implies U \to V$ igaz, hiszen nincs olyan két sor, ami U-n megegyezik.

Bizonyítás (folyt.)



Tehát r két soros reláció, $X^+(F)$-en megegyezik a két sor, a többin különbözik.

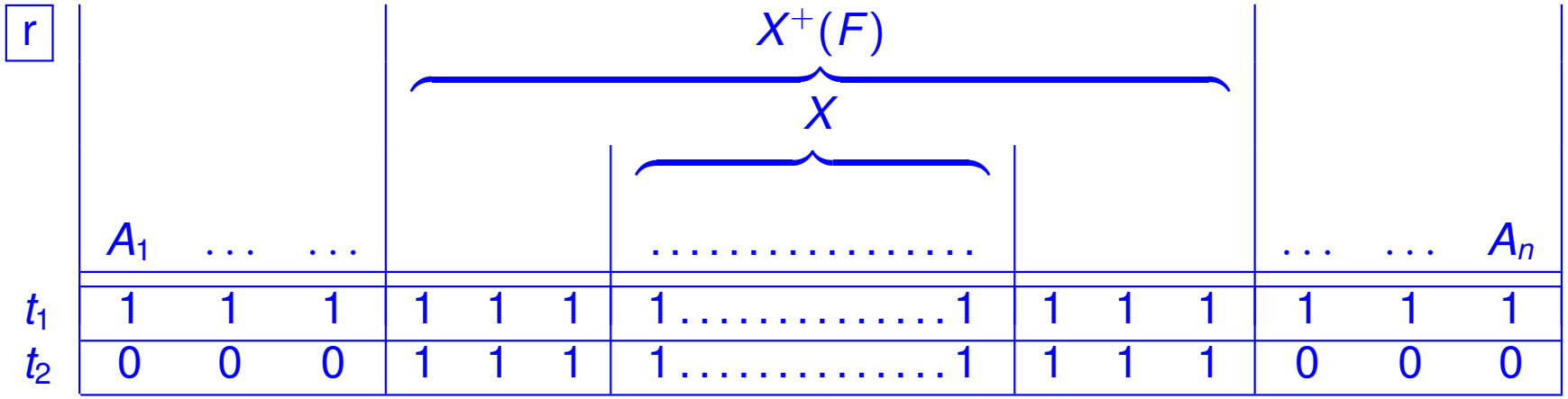
Állítás

r-ben teljesülnek F függései.

Bizonyítás.

Legyen $U \to V \in F$.
Ha $U \not\subseteq X^+(F) \implies U \to V$ igaz, hiszen nincs olyan két sor, ami U-n megegyezik.
Ha $U \subseteq X^+(F)$, akkor lemma miatt $F \vdash X \to U$.
Tranzitivitás miatt $F \vdash X \to V$. Lemma $\implies V \subseteq X^+(F)$, V-n megegyezik a két sor.
$\surd$

Bizonyítás (folyt.)



Tehát r két soros reláció, $X^+(F)$-en megegyezik a két sor, a többin különbözik.

Állítás

r-ben teljesülnek F függései.

Bizonyítás.

Legyen $U \to V \in F$.
Ha $U \not\subseteq X^+(F) \implies U \to V$ igaz, hiszen nincs olyan két sor, ami U-n megegyezik.
Ha $U \subseteq X^+(F)$, akkor lemma miatt $F \vdash X \to U$.
Tranzitivitás miatt $F \vdash X \to V$. Lemma $\implies V \subseteq X^+(F)$, V-n megegyezik a két sor.
$\surd$ □

Bizonyítás (folyt.)

Állítás

r-ben nem igaz $X \to Y$.

Bizonyítás (folyt.)

Állítás

r-ben nem igaz $X \to Y$.

Bizonyítás.

Mivel $F \not\models X \to Y$, ezért a fontos lemma miatt $Y \not\subseteq X^+(F)$, azaz Y kilóg $X^+(F)$-ből, abból a részből, ahol a két sor egyenlő.

Bizonyítás (folyt.)

Állítás

r-ben nem igaz $X \to Y$.

Bizonyítás.

Mivel $F \not\models X \to Y$, ezért a fontos lemma miatt $Y \not\subseteq X^+(F)$, azaz Y kilóg $X^+(F)$-ből, abból a részből, ahol a két sor egyenlő.
Vagyis a két sor egyenlő X-en, de nem egyenlő Y-on, így $X \to Y$ nem igaz r-ben.

Bizonyítás (folyt.)

Állítás

r-ben nem igaz $X \to Y$.

Bizonyítás.

Mivel $F \not\vdash X \to Y$, ezért a fontos lemma miatt $Y \not\subseteq X^+(F)$, azaz Y kilóg $X^+(F)$-ből, abból a részből, ahol a két sor egyenlő.
Vagyis a két sor egyenlő X-en, de nem egyenlő Y-on, így $X \to Y$ nem igaz r-ben. □

Tehát r tényleg olyan, hogy benne F minden függése fennáll, de $X \to Y$ nem, ami bizonyítja, hogy $F \not\models X \to Y$.

Bizonyítás (folyt.)

Állítás

r-ben nem igaz $X \to Y$.

Bizonyítás.

Mivel $F \nvdash X \to Y$, ezért a fontos lemma miatt $Y \not\subseteq X^+(F)$, azaz Y kilóg $X^+(F)$-ből, abból a részből, ahol a két sor egyenlő.
Vagyis a két sor egyenlő X-en, de nem egyenlő Y-on, így $X \to Y$ nem igaz r-ben. □

Tehát r tényleg olyan, hogy benne F minden függése fennáll, de $X \to Y$ nem, ami bizonyítja, hogy $F \not\models X \to Y$.

Következmény

$\vdash$ *és* $\models$ *felcserélhető.*

Kulcs

Definíció

$X \subseteq R$ ***szuperkulcsa*** *az (R, F) sémának, ha $F \vdash X \to R$. Másképpen, ha $R = X^+(F)$.*

Kulcs

Definíció

*$X \subseteq R$ **szuperkulcsa** az (R, F) sémának, ha $F \vdash X \to R$. Másképpen, ha $R = X^+(F)$.*
*$X \subseteq R$ **kulcsa** az (R, F) sémának, ha szuperkulcs és nincs olyan valódi részhalmaza, ami szuperkulcs.*

Kulcs

Definíció

*$X \subseteq R$ **szuperkulcsa** az (R, F) sémának, ha $F \vdash X \to R$. Másképpen, ha $R = X^+(F)$.*
*$X \subseteq R$ **kulcsa** az (R, F) sémának, ha szuperkulcs és nincs olyan valódi részhalmaza, ami szuperkulcs.*

Példa:
$F = \{\text{TERMELŐ, TERMÉKNÉV} \to \text{ÁR; TERMELŐ} \to \text{CÍM}\}$
$X = \text{TERMELŐ, TERMÉKNÉV}$

Kulcs

Definíció

*$X \subseteq R$ **szuperkulcsa** az (R, F) sémának, ha $F \vdash X \to R$. Másképpen, ha $R = X^+(F)$.*
*$X \subseteq R$ **kulcsa** az (R, F) sémának, ha szuperkulcs és nincs olyan valódi részhalmaza, ami szuperkulcs.*

Példa:
$F = \{\text{TERMELŐ, TERMÉKNÉV} \to \text{ÁR; TERMELŐ} \to \text{CÍM}\}$
$X = \text{TERMELŐ, TERMÉKNÉV}$
$\Longrightarrow \; X^+(F) = \text{TERMELŐ, TERMÉKNÉV, ÁR, CÍM}$

Kulcs

Definíció

*$X \subseteq R$ **szuperkulcsa** az (R, F) sémának, ha $F \vdash X \to R$. Másképpen, ha $R = X^+(F)$.*
*$X \subseteq R$ **kulcsa** az (R, F) sémának, ha szuperkulcs és nincs olyan valódi részhalmaza, ami szuperkulcs.*

Példa:
$F = \{\text{TERMELŐ, TERMÉKNÉV} \to \text{ÁR; TERMELŐ} \to \text{CÍM}\}$
$X = \text{TERMELŐ, TERMÉKNÉV}$
$\Longrightarrow \; X^+(F) = \text{TERMELŐ, TERMÉKNÉV, ÁR, CÍM}$
$\text{TERMELŐ}^+(F) = \text{TERMELŐ, CÍM}$
$\text{TERMÉKNÉV}^+(F) = \text{TERMÉKNÉV}$

Kulcs

Definíció

*$X \subseteq R$ **szuperkulcsa** az (R, F) sémának, ha $F \vdash X \to R$. Másképpen, ha $R = X^+(F)$.*
*$X \subseteq R$ **kulcsa** az (R, F) sémának, ha szuperkulcs és nincs olyan valódi részhalmaza, ami szuperkulcs.*

Példa:
$F = \{\text{TERMELŐ, TERMÉKNÉV} \to \text{ÁR; TERMELŐ} \to \text{CÍM}\}$
$X = \text{TERMELŐ, TERMÉKNÉV}$
$\Longrightarrow X^+(F) = \text{TERMELŐ, TERMÉKNÉV, ÁR, CÍM}$
$\text{TERMELŐ}^+(F) = \text{TERMELŐ, CÍM}$
$\text{TERMÉKNÉV}^+(F) = \text{TERMÉKNÉV}$
$\Longrightarrow X$ kulcs

$X^+(F)$ kiszámítása

Algoritmus:

$X_0 = X$,

$\vdots$

$X_i = \ldots$,

$X^+(F)$ kiszámítása

Algoritmus:

$X_0 = X$,

$\vdots$

$X_i = \ldots$,

$X_{i+1} = X_i \cup \{A \in R \mid \text{van olyan } U \to V \in F, \text{hogy } U \subseteq X_i \text{ és } A \in V\}$

$X^+(F)$ kiszámítása

Algoritmus:

$X_0 = X$,

$\vdots$

$X_i = \ldots$,

$X_{i+1} = X_i \cup \{A \in R \mid \text{van olyan } U \to V \in F, \text{ hogy } U \subseteq X_i \text{ és } A \in V\}$,

$\vdots$

$X^+(F) = X_{\text{utolsó}}$, (amikor már nem nő)

$X^+(F)$ kiszámítása

Algoritmus:

$X_0 = X$,

$\vdots$

$X_i = \ldots$,

$X_{i+1} = X_i \cup \{A \in R \mid \text{van olyan } U \to V \in F, \text{ hogy } U \subseteq X_i \text{ és } A \in V\}$,

$\vdots$

$X^+(F) = X_{\text{utolsó}}$, (amikor már nem nő)

Állítás

$X_{utolsó} \subseteq X^+(F)$ *(azaz, a fontos lemma miatt* $F \vdash X \to X_{utolsó}$*)*

$X^+(F)$ kiszámítása

Algoritmus:

$X_0 = X$,

$\vdots$

$X_i = \ldots$,

$X_{i+1} = X_i \cup \{A \in R \mid \text{van olyan } U \to V \in F, \text{ hogy } U \subseteq X_i \text{ és } A \in V\}$,

$\vdots$

$X^+(F) = X_{\text{utolsó}}$, (amikor már nem nő)

Állítás

$X_{utolsó} \subseteq X^+(F)$ *(azaz, a fontos lemma miatt* $F \vdash X \to X_{utolsó}$*)*

Bizonyítás.

Indukcióval i*-re belátjuk, hogy* $F \vdash X \to X_i$*, innen már következik az állítás.*

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.
$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$
8. $F \vdash X \to X_{i+1}$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$
8. $F \vdash X \to X_{i+1}$

$\Longrightarrow F \vdash X \to X_{\text{utolsó}} \Longrightarrow$ (Lemma) $X_{\text{utolsó}} \subseteq X^+(F)$. $\surd$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.

$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$
8. $F \vdash X \to X_{i+1}$

$\Longrightarrow F \vdash X \to X_{\text{utolsó}} \Longrightarrow$ (Lemma) $X_{\text{utolsó}} \subseteq X^+(F)$. $\surd$ □

Példa:

$R(A, B, C, D), \quad F = \{A \to B,\ B \to C,\ BC \to D\}, \qquad A^+(F) = ?$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.
$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$
8. $F \vdash X \to X_{i+1}$

$\Longrightarrow F \vdash X \to X_{\text{utolsó}} \Longrightarrow$ (Lemma) $X_{\text{utolsó}} \subseteq X^+(F)$. $\surd$ □

Példa:
$R(A, B, C, D),\quad F = \{A \to B,\ B \to C,\ BC \to D\},\qquad A^+(F) = ?$
$X_0 = \{A\},$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.
$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$
8. $F \vdash X \to X_{i+1}$

$\Longrightarrow F \vdash X \to X_{\text{utolsó}} \Longrightarrow$ (Lemma) $X_{\text{utolsó}} \subseteq X^+(F)$. $\surd$ □

Példa:
$R(A, B, C, D),\quad F = \{A \to B,\ B \to C,\ BC \to D\},\qquad A^+(F) = ?$
$X_0 = \{A\},\quad X_1 = \{A, B\},$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.
$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$
8. $F \vdash X \to X_{i+1}$

$\Longrightarrow F \vdash X \to X_{\text{utolsó}} \Longrightarrow$ (Lemma) $X_{\text{utolsó}} \subseteq X^+(F)$. $\surd$ □

Példa:
$R(A, B, C, D),\quad F = \{A \to B,\ B \to C,\ BC \to D\},\qquad A^+(F) = ?$
$X_0 = \{A\},\quad X_1 = \{A, B\},\quad X_2 = \{A, B, C\},$

Bizonyítás (folyt.)

Bizonyítás.

$i = 0$: $F \vdash X \to X_0 = X$, reflexivitás.
$i \rightsquigarrow i + 1$

1. $U \to V \in F,\ U \subseteq X_i,\ A \in V$
2. $X_i \to U$, reflexivitás
3. $X \to U$, tranzitivitás + indukciós felt. $F \vdash X \to X_i$ + (2. sor)
4. $X \to V$, tranzitivitás (1. és 3. sor)
5. $V \to A$, reflexivitás, (1. sor)
6. $X \to A$, tranzitivitás, (4. és 5. sor)
7. $F \vdash X \to A$, minden $A \in X_{i+1}$
8. $F \vdash X \to X_{i+1}$

$\Longrightarrow F \vdash X \to X_{\text{utolsó}} \Longrightarrow$ (Lemma) $X_{\text{utolsó}} \subseteq X^+(F)$. $\surd$ □

Példa:
$R(A, B, C, D),\quad F = \{A \to B,\ B \to C,\ BC \to D\},\qquad A^+(F) = ?$
$X_0 = \{A\},\quad X_1 = \{A, B\},\quad X_2 = \{A, B, C\},\quad X_3 = \{A, B, C, D\} = X_{\text{utolsó}}$

Bizonyítás másik iránya

Állítás

$X^+(F) \subseteq X_{utolsó}$

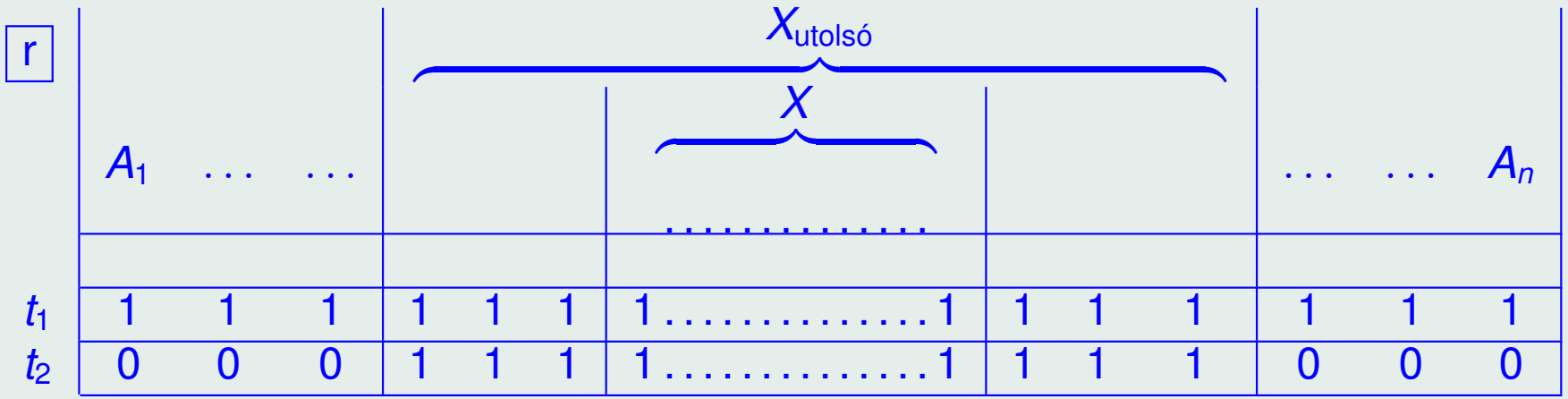
Bizonyítás másik iránya

Állítás

$X^+(F) \subseteq X_{utolsó}$

Bizonyítás.

Tekintsük az alábbi kétsoros r relációt: a két sor $X_{\text{utolsó}}$-n egyenlő, azon kívül eltérnek.



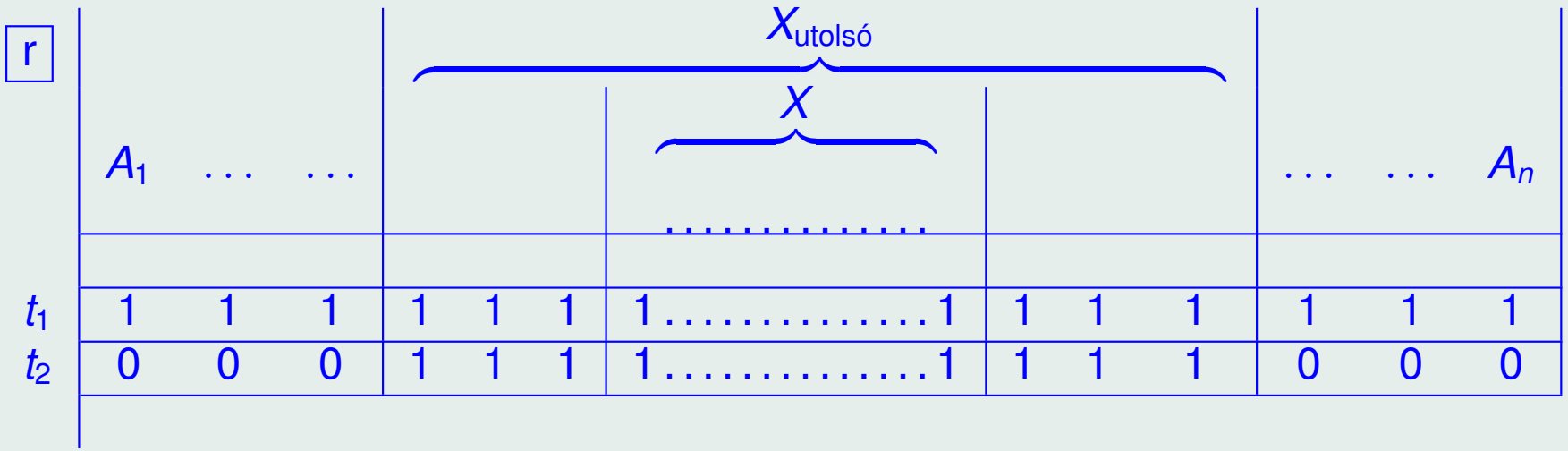
Bizonyítás másik iránya

Állítás

$X^+(F) \subseteq X_{utolsó}$

Bizonyítás.

Tekintsük az alábbi kétsoros r relációt: a két sor $X_{\text{utolsó}}$-n egyenlő, azon kívül eltérnek.



r-ben egyrészt minden F-beli függés teljesül. (Bizonyítás, hasonlóan, mint a teljességi tételnél csak most az kell, hogy ha egy $W \to S$ függés esetén $W \subseteq X_{\text{utolsó}}$, akkor $S \subseteq X_{\text{utolsó}}$ is igaz, az algoritmus működése miatt.)

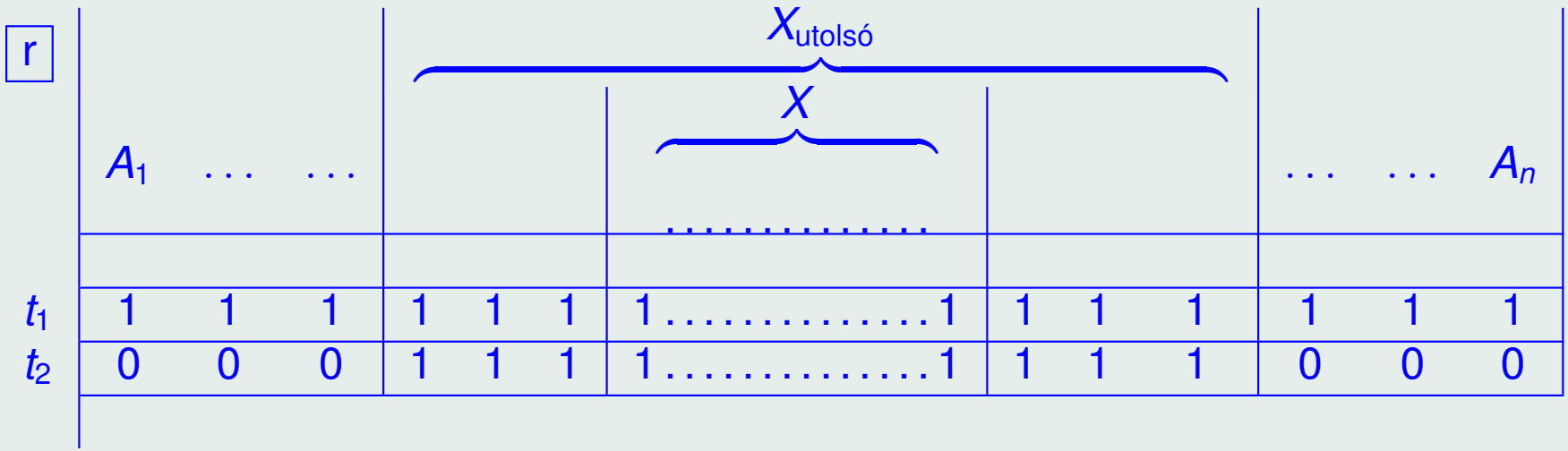
Bizonyítás másik iránya

Állítás

$X^+(F) \subseteq X_{utolsó}$

Bizonyítás.

Tekintsük az alábbi kétsoros r relációt: a két sor $X_{\text{utolsó}}$-n egyenlő, azon kívül eltérnek.



r-ben egyrészt minden F-beli függés teljesül. (Bizonyítás, hasonlóan, mint a teljességi tételnél csak most az kell, hogy ha egy $W \to S$ függés esetén $W \subseteq X_{\text{utolsó}}$, akkor $S \subseteq X_{\text{utolsó}}$ is igaz, az algoritmus működése miatt.)
r segítségével azt látjuk be, hogy ha $A \notin X_{\text{utolsó}}$, akkor $A \notin X^+(F)$, ami éppen a kívánt állítás.

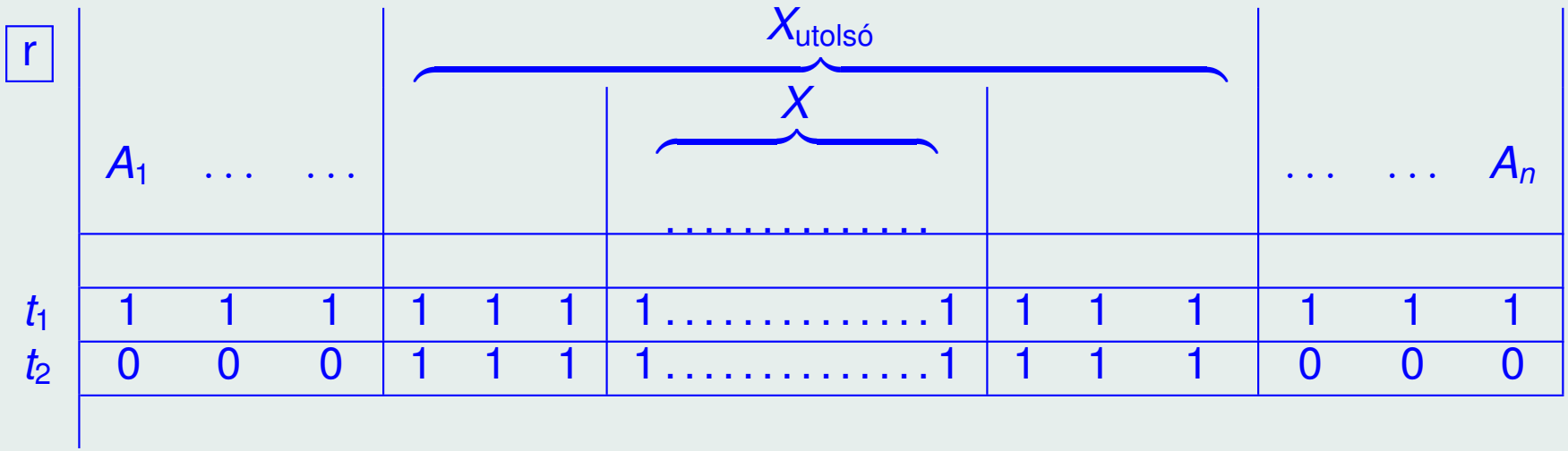
Bizonyítás másik iránya

Állítás

$X^+(F) \subseteq X_{utolsó}$

Bizonyítás.

Tekintsük az alábbi kétsoros r relációt: a két sor $X_{\text{utolsó}}$-n egyenlő, azon kívül eltérnek.



r-ben egyrészt minden F-beli függés teljesül. (Bizonyítás, hasonlóan, mint a teljességi tételnél csak most az kell, hogy ha egy $W \to S$ függés esetén $W \subseteq X_{\text{utolsó}}$, akkor $S \subseteq X_{\text{utolsó}}$ is igaz, az algoritmus működése miatt.)
r segítségével azt látjuk be, hogy ha $A \notin X_{\text{utolsó}}$, akkor $A \notin X^+(F)$, ami éppen a kívánt állítás.

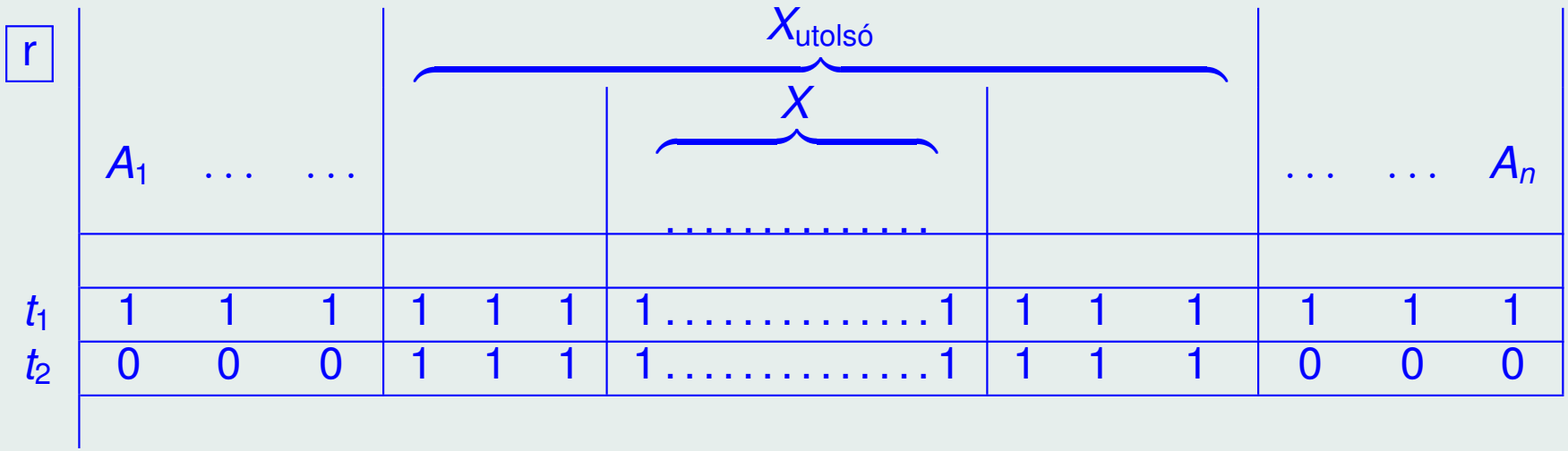
Bizonyítás másik iránya

Állítás

$X^+(F) \subseteq X_{utolsó}$

Bizonyítás.

Tekintsük az alábbi kétsoros r relációt: a két sor $X_{\text{utolsó}}$-n egyenlő, azon kívül eltérnek.



r-ben egyrészt minden F-beli függés teljesül. (Bizonyítás, hasonlóan, mint a teljességi tételnél csak most az kell, hogy ha egy $W \to S$ függés esetén $W \subseteq X_{\text{utolsó}}$, akkor $S \subseteq X_{\text{utolsó}}$ is igaz, az algoritmus működése miatt.)
r segítségével azt látjuk be, hogy ha $A \notin X_{\text{utolsó}}$, akkor $A \notin X^+(F)$, ami éppen a kívánt állítás.

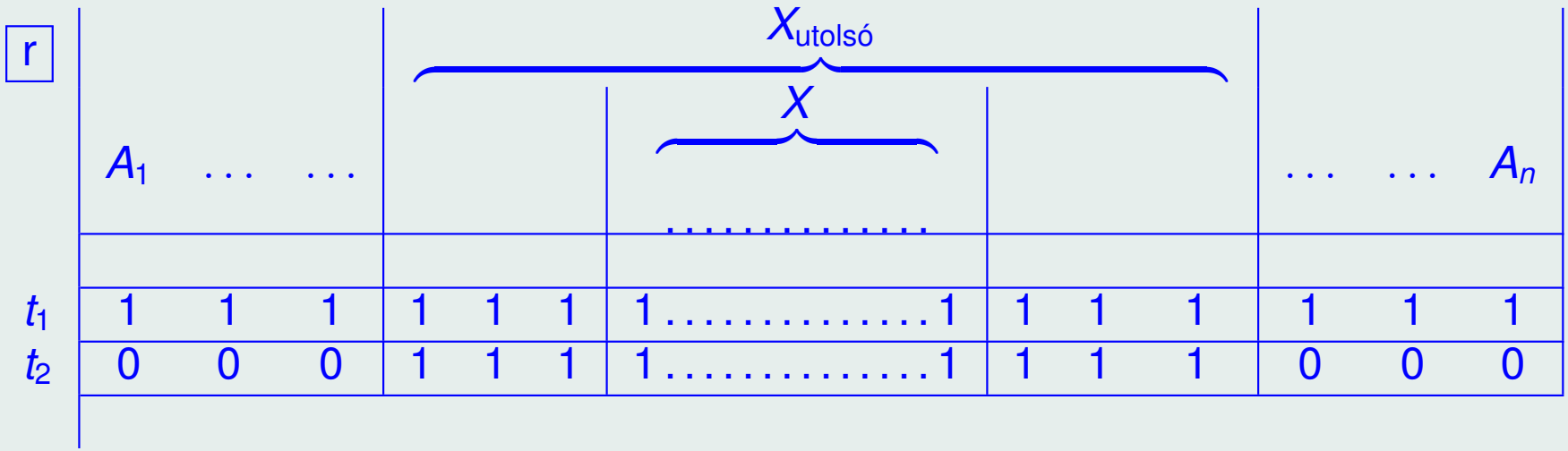
Bizonyítás másik iránya

Állítás

$X^+(F) \subseteq X_{utolsó}$

Bizonyítás.

Tekintsük az alábbi kétsoros r relációt: a két sor $X_{\text{utolsó}}$-n egyenlő, azon kívül eltérnek.



r-ben egyrészt minden F-beli függés teljesül. (Bizonyítás, hasonlóan, mint a teljességi tételnél csak most az kell, hogy ha egy $W \to S$ függés esetén $W \subseteq X_{\text{utolsó}}$, akkor $S \subseteq X_{\text{utolsó}}$ is igaz, az algoritmus működése miatt.)
r segítségével azt látjuk be, hogy ha $A \notin X_{\text{utolsó}}$, akkor $A \notin X^+(F)$, ami éppen a kívánt állítás.

Következmények

Következmény

$X^+(F) = X_{utolsó}$, azaz tényleg jó az algoritmus.

Következmények

Következmény

$X^+(F) = X_{\text{utolsó}}$, azaz tényleg jó az algoritmus.

Következmény

Adott X-ről el lehet dönteni, hogy (szuper)kulcs-e.

Következmények

Következmény

$X^+(F) = X_{\text{utolsó}}$, azaz tényleg jó az algoritmus.

Következmény

Adott X-ről el lehet dönteni, hogy (szuper)kulcs-e.

Megnézzük, hogy $X^+(F) = R$ igaz-e. Ha igen, akkor szuperkulcs. Ha minden $X - A$-ra már nem szuperkulcsot kapunk, akkor X kulcs.

Következmények

Következmény

$X^+(F) = X_{utolsó}$, azaz tényleg jó az algoritmus.

Következmény

Adott X-ről el lehet dönteni, hogy (szuper)kulcs-e.

Megnézzük, hogy $X^+(F) = R$ igaz-e. Ha igen, akkor szuperkulcs. Ha minden $X - A$-ra már nem szuperkulcsot kapunk, akkor X kulcs.
Gyorsan implementálható.

Adatbázisok elmélete

Hűséges felbontás, normálformák

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

13. előadás

Felbontások

Cél: *Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontás előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.*

Felbontások

Cél: *Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontás előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.*

Definíció

$\rho = (R_1, \ldots, R_k)$ az (R, F) séma felbontása, ha $R_i \subseteq R$ és $\cup_{i=1}^{k} R_i = R$.
Ha r egy (R, F) sémára illeszkedő reláció, akkor legyen $r_i = \pi_{R_i}(r)$ és

$$m_\rho(r) := r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$$

Felbontások

Cél: *Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontás előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.*

Definíció

$\rho = (R_1, \ldots, R_k)$ az (R, F) séma felbontása, ha $R_i \subseteq R$ és $\cup_{i=1}^{k} R_i = R$.
Ha r egy (R, F) sémára illeszkedő reláció, akkor legyen $r_i = \pi_{R_i}(r)$ és

$$m_\rho(r) := r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$$

(Megj.: $\bowtie$ asszociatív, így nem kell a zárójelezéssel vesződni)

Felbontások

Cél: *Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontás előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.*

Definíció

$\rho = (R_1, \ldots, R_k)$ az (R, F) séma felbontása, ha $R_i \subseteq R$ és $\cup_{i=1}^{k} R_i = R$.
Ha r egy (R, F) sémára illeszkedő reláció, akkor legyen $r_i = \pi_{R_i}(r)$ és

$$m_\rho(r) := r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$$

(Megj.: $\bowtie$ asszociatív, így nem kell a zárójelezéssel vesződni)

Kérdés: mikor nyerhető vissza az infó a felbontásból? Mi általában r és $m_\rho(r)$ viszonya?

Felbontások

Cél: *Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontás előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.*

Definíció

$\rho = (R_1, \ldots, R_k)$ az (R, F) séma felbontása, ha $R_i \subseteq R$ és $\cup_{i=1}^{k} R_i = R$.
Ha r egy (R, F) sémára illeszkedő reláció, akkor legyen $r_i = \pi_{R_i}(r)$ és

$$m_\rho(r) := r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$$

(Megj.: $\bowtie$ asszociatív, így nem kell a zárójelezéssel vesződni)

Kérdés: mikor nyerhető vissza az infó a felbontásból? Mi általában r és $m_\rho(r)$ viszonya?

Tétel

(i) $r \subseteq m_\rho(r)$

Felbontások

Cél: *Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontás előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.*

Definíció

$\rho = (R_1, \ldots, R_k)$ az (R, F) séma felbontása, ha $R_i \subseteq R$ és $\cup_{i=1}^{k} R_i = R$.
Ha r egy (R, F) sémára illeszkedő reláció, akkor legyen $r_i = \pi_{R_i}(r)$ és

$$m_\rho(r) := r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$$

(Megj.: $\bowtie$ asszociatív, így nem kell a zárójelezéssel vesződni)

Kérdés: mikor nyerhető vissza az infó a felbontásból? Mi általában r és $m_\rho(r)$ viszonya?

Tétel

(i) $r \subseteq m_\rho(r)$

(ii) $r_i = \pi_{R_i}(m_\rho(r))$

Felbontások

Cél: *Adott (R, F) sémából anomáliát nem tartalmazó olyan felbontás előállítása, amiből ugyanaz az információ nyerhető, mint az eredetiből.*

Definíció

$\rho = (R_1, \ldots, R_k)$ az (R, F) séma felbontása, ha $R_i \subseteq R$ és $\cup_{i=1}^{k} R_i = R$.
Ha r egy (R, F) sémára illeszkedő reláció, akkor legyen $r_i = \pi_{R_i}(r)$ és

$$m_\rho(r) := r_1 \bowtie r_2 \bowtie \cdots \bowtie r_k$$

(Megj.: $\bowtie$ asszociatív, így nem kell a zárójelezéssel vesződni)

Kérdés: mikor nyerhető vissza az infó a felbontásból? Mi általában r és $m_\rho(r)$ viszonya?

Tétel

(i) $r \subseteq m_\rho(r)$

(ii) $r_i = \pi_{R_i}(m_\rho(r))$

(iii) $m_\rho(m_\rho(r)) = m_\rho(r)$

Bizonyítás

Bizonyítás.

Ha t egy sor, akkor $\pi_{R_i}(t)$ helyett $t[R_i]$-t írunk.

(i) $r \subseteq m_\rho(r)$:
Ha t egy sor r-ben, akkor t minden vetülete benne van a megfelelő $t[R_i]$-ben, ezek össze is illenek, így $m_\rho(r)$-ben is szerepelni fog t.

Bizonyítás

Bizonyítás.

Ha t egy sor, akkor $\pi_{R_i}(t)$ helyett $t[R_i]$-t írunk.

(i) $r \subseteq m_\rho(r)$:
Ha t egy sor r-ben, akkor t minden vetülete benne van a megfelelő $t[R_i]$-ben, ezek össze is illenek, így $m_\rho(r)$-ben is szerepelni fog t.

(ii) $r_i = \pi_{R_i}(m_\rho(r))$:
$r \subseteq m_\rho(r) \implies r_i = \pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(r))$.

Bizonyítás

Bizonyítás.

Ha t egy sor, akkor $\pi_{R_i}(t)$ helyett $t[R_i]$-t írunk.

(i) $r \subseteq m_\rho(r)$:
Ha t egy sor r-ben, akkor t minden vetülete benne van a megfelelő $t[R_i]$-ben, ezek össze is illenek, így $m_\rho(r)$-ben is szerepelni fog t.

(ii) $r_i = \pi_{R_i}(m_\rho(r))$:
$r \subseteq m_\rho(r) \implies r_i = \pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(r))$.
Ha $t \in m_\rho(r)$, akkor ez természetes illesztéssel jött létre, r_i-beli sorokból, így levetítve R_i-re épp r_i egy sorát kapjuk.

Bizonyítás

Bizonyítás.

Ha t egy sor, akkor $\pi_{R_i}(t)$ helyett $t[R_i]$-t írunk.

(i) $r \subseteq m_\rho(r)$:
Ha t egy sor r-ben, akkor t minden vetülete benne van a megfelelő $t[R_i]$-ben, ezek össze is illenek, így $m_\rho(r)$-ben is szerepelni fog t.

(ii) $r_i = \pi_{R_i}(m_\rho(r))$:
$r \subseteq m_\rho(r) \implies r_i = \pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(r))$.
Ha $t \in m_\rho(r)$, akkor ez természetes illesztéssel jött létre, r_i-beli sorokból, így levetítve R_i-re épp r_i egy sorát kapjuk.

(iii) $m_\rho(m_\rho(r)) = m_\rho(r)$:
$m_\rho(r) = \bowtie_{i=1}^{k} r_i = \bowtie_{i=1}^{k} \pi_{R_i}(r)$

Bizonyítás

Bizonyítás.

Ha t egy sor, akkor $\pi_{R_i}(t)$ helyett $t[R_i]$-t írunk.

(i) $r \subseteq m_\rho(r)$:
Ha t egy sor r-ben, akkor t minden vetülete benne van a megfelelő $t[R_i]$-ben, ezek össze is illenek, így $m_\rho(r)$-ben is szerepelni fog t.

(ii) $r_i = \pi_{R_i}(m_\rho(r))$:
$r \subseteq m_\rho(r) \implies r_i = \pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(r))$.
Ha $t \in m_\rho(r)$, akkor ez természetes illesztéssel jött létre, r_i-beli sorokból, így levetítve R_i-re épp r_i egy sorát kapjuk.

(iii) $m_\rho(m_\rho(r)) = m_\rho(r)$:
$m_\rho(r) = \bowtie_{i=1}^{k} r_i = \bowtie_{i=1}^{k} \pi_{R_i}(r)$
$m_\rho(m_\rho(r)) = \bowtie_{i=1}^{k} \pi_{R_i}(m_\rho(r)) \overset{(ii)}{=} \bowtie_{i=1}^{k} r_i = m_\rho(r)$

Bizonyítás

Bizonyítás.

Ha t egy sor, akkor $\pi_{R_i}(t)$ helyett $t[R_i]$-t írunk.

(i) $r \subseteq m_\rho(r)$:
Ha t egy sor r-ben, akkor t minden vetülete benne van a megfelelő $t[R_i]$-ben, ezek össze is illenek, így $m_\rho(r)$-ben is szerepelni fog t.

(ii) $r_i = \pi_{R_i}(m_\rho(r))$:
$r \subseteq m_\rho(r) \implies r_i = \pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(r))$.
Ha $t \in m_\rho(r)$, akkor ez természetes illesztéssel jött létre, r_i-beli sorokból, így levetítve R_i-re épp r_i egy sorát kapjuk.

(iii) $m_\rho(m_\rho(r)) = m_\rho(r)$:
$m_\rho(r) = \bowtie_{i=1}^{k} r_i = \bowtie_{i=1}^{k} \pi_{R_i}(r)$
$m_\rho(m_\rho(r)) = \bowtie_{i=1}^{k} \pi_{R_i}(m_\rho(r)) \overset{(ii)}{=} \bowtie_{i=1}^{k} r_i = m_\rho(r)$

□

Megjegyzés: (i) szerint a szétszedés és összerakás után vagy pont r-t kapom meg, vagy többet kapok, kevesebb sor nem lehet. Ha $r \neq m_\rho(r)$, akkor ez nem egy túl hasznos felbontás.

Megjegyzés: (i) szerint a szétszedés és összerakás után vagy pont r-t kapom meg, vagy többet kapok, kevesebb sor nem lehet. Ha $r \neq m_\rho(r)$, akkor ez nem egy túl hasznos felbontás.
De ennél több is igaz: ebben az esetben teljesen reménytelen a felbontásból visszaszerezni r-t: mivel (ii) szerint r és $m_\rho(r)$ (függőleges) vetületei ugyanazok, ezért ha $r \neq m_\rho(r)$, akkor *van két olyan reláció (r és $m_\rho(r)$), aminek a vetületei ugyanazok $\Longrightarrow$ a vetületekből nem lehet visszaállítani r-et (nem lehet eldönteni, hogy r vagy $m_\rho(r)$ volt).*

Megjegyzés: (i) szerint a szétszedés és összerakás után vagy pont r-t kapom meg, vagy többet kapok, kevesebb sor nem lehet. Ha $r \neq m_\rho(r)$, akkor ez nem egy túl hasznos felbontás.
De ennél több is igaz: ebben az esetben teljesen reménytelen a felbontásból visszaszerezni r-t: mivel (ii) szerint r és $m_\rho(r)$ (függőleges) vetületei ugyanazok, ezért ha $r \neq m_\rho(r)$, akkor *van két olyan reláció (r és $m_\rho(r)$), aminek a vetületei ugyanazok $\Longrightarrow$ a vetületekből nem lehet visszaállítani r-et (nem lehet eldönteni, hogy r vagy $m_\rho(r)$ volt).*

Következmény: ha $r \neq m_\rho(r)$, akkor sehogyan se lehet visszahozni r-t a vetületekből.

Hűséges felbontás

Tehát az a kérdés, hogy mik azok a felbontásai egy (R, F) sémának, amik esetén tetszőleges (R, F)-re illeszkedő r relációra $r = m_\rho(r)$

Hűséges felbontás

Tehát az a kérdés, hogy mik azok a felbontásai egy (R, F) sémának, amik esetén tetszőleges (R, F)-re illeszkedő r relációra $r = m_\rho(r)$

Definíció

*Adott (R, F). Ennek ρ felbontása **hűséges (veszteségmentes, lossless)**, ha minden (R, F)-re illeszkedő r relációra $r = m_\rho(r)$.*

Példa: Legyen (R, F) a következő: $R(A, B, C)$, $F = \{C \to A\}$ és legyen r az alábbi reláció.

r	A	B	C
	a	c	e
	a	d	f
	b	c	g
	b	d	h

Hűséges felbontás

Tehát az a kérdés, hogy mik azok a felbontásai egy (R, F) sémának, amik esetén tetszőleges (R, F)-re illeszkedő r relációra $r = m_\rho(r)$

Definíció

*Adott (R, F). Ennek ρ felbontása **hűséges (veszteségmentes, lossless)**, ha minden (R, F)-re illeszkedő r relációra $r = m_\rho(r)$.*

Példa: Legyen (R, F) a következő: $R(A, B, C)$, $F = \{C \to A\}$ és legyen r az alábbi reláció.

r	A	B	C
	a	c	e
	a	d	f
	b	c	g
	b	d	h

s	A	B
	a	c
	a	d
	b	c
	b	d

t	B	C
	c	e
	d	f
	c	g
	d	h

Hűséges felbontás

Tehát az a kérdés, hogy mik azok a felbontásai egy (R, F) sémának, amik esetén tetszőleges (R, F)-re illeszkedő r relációra $r = m_\rho(r)$

Definíció

*Adott (R, F). Ennek ρ felbontása **hűséges (veszteségmentes, lossless)**, ha minden (R, F)-re illeszkedő r relációra $r = m_\rho(r)$.*

Példa: Legyen (R, F) a következő: $R(A, B, C)$, $F = \{C \to A\}$ és legyen r az alábbi reláció.

r	A	B	C
	a	c	e
	a	d	f
	b	c	g
	b	d	h

s	A	B
	a	c
	a	d
	b	c
	b	d

t	B	C
	c	e
	d	f
	c	g
	d	h

$s \bowtie t$	A	B	C
	a	c	e
	a	c	g
	a	d	f
	a	d	h
	b	c	e
	b	c	g
	b	d	f
	b	d	h

Hűséges felbontás

Tehát az a kérdés, hogy mik azok a felbontásai egy (R, F) sémának, amik esetén tetszőleges (R, F)-re illeszkedő r relációra $r = m_\rho(r)$

Definíció

*Adott (R, F). Ennek ρ felbontása **hűséges (veszteségmentes, lossless)**, ha minden (R, F)-re illeszkedő r relációra $r = m_\rho(r)$.*

Példa: Legyen (R, F) a következő: $R(A, B, C)$, $F = \{C \to A\}$ és legyen r az alábbi reláció.

r	A	B	C
	a	c	e
	a	d	f
	b	c	g
	b	d	h

s	A	B
	a	c
	a	d
	b	c
	b	d

t	B	C
	c	e
	d	f
	c	g
	d	h

$s \bowtie t$	A	B	C
	a	c	e
	a	c	g
	a	d	f
	a	d	h
	b	c	e
	b	c	g
	b	d	f
	b	d	h

Ez a példa mutatja, hogy $r \neq s(A, B) \bowtie t(B, C)$, azaz ez a felbontás nem hűséges.

Hűséges felbontás

Tehát az a kérdés, hogy mik azok a felbontásai egy (R, F) sémának, amik esetén tetszőleges (R, F)-re illeszkedő r relációra $r = m_\rho(r)$

Definíció

*Adott (R, F). Ennek ρ felbontása **hűséges (veszteségmentes, lossless)**, ha minden (R, F)-re illeszkedő r relációra $r = m_\rho(r)$.*

Példa: Legyen (R, F) a következő: $R(A, B, C)$, $F = \{C \to A\}$ és legyen r az alábbi reláció.

r	A	B	C
	a	c	e
	a	d	f
	b	c	g
	b	d	h

s	A	B
	a	c
	a	d
	b	c
	b	d

t	B	C
	c	e
	d	f
	c	g
	d	h

$s \bowtie t$	A	B	C
	a	c	e
	a	c	g
	a	d	f
	a	d	h
	b	c	e
	b	c	g
	b	d	f
	b	d	h

Ez a példa mutatja, hogy $r \neq s(A, B) \bowtie t(B, C)$, azaz ez a felbontás nem hűséges.
De $r = s'(A, C) \bowtie t'(B, C)$, majd látjuk.

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\iff$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\Longleftrightarrow$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Példa: R(TNÉV, TERMELŐ, ÁR, CÍM)

$F = \{\text{TERMELŐ} \to \text{CÍM};\ \text{TNÉV, TERMELŐ} \to \text{ÁR}\}$

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\Longleftrightarrow$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Példa: R(TNÉV, TERMELŐ, ÁR, CÍM)

$F = \{\text{TERMELŐ} \to \text{CÍM};\ \text{TNÉV, TERMELŐ} \to \text{ÁR}\}$

$\rho = (\text{TERMELŐ, CÍM} ; \text{TNÉV, TERMELŐ, ÁR})$

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\Longleftrightarrow$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Példa: R(TNÉV, TERMELŐ, ÁR, CÍM)

$F = \{\text{TERMELŐ} \to \text{CÍM};\ \text{TNÉV, TERMELŐ} \to \text{ÁR}\}$

$\rho = (\text{TERMELŐ, CÍM} ; \text{TNÉV, TERMELŐ, ÁR})$

$R_1 \cap R_2 = \{\text{TERMELŐ}\}, \quad R_1 \setminus R_2 = \{\text{CÍM}\}, \quad R_2 \setminus R_1 = \{\text{TNÉV, ÁR}\}$

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\Longleftrightarrow$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Példa: R(TNÉV, TERMELŐ, ÁR, CÍM)

$F = \{\text{TERMELŐ} \to \text{CÍM};\ \text{TNÉV, TERMELŐ} \to \text{ÁR}\}$

$\rho = (\text{TERMELŐ, CÍM} ; \text{TNÉV, TERMELŐ, ÁR})$

$R_1 \cap R_2 = \{\text{TERMELŐ}\}, \quad R_1 \setminus R_2 = \{\text{CÍM}\}, \quad R_2 \setminus R_1 = \{\text{TNÉV, ÁR}\}$

$\Longrightarrow (\text{TERMELŐ})^+(F) = \{\text{TERMELŐ, CÍM}\} \supseteq R_1 \setminus R_2 \Longrightarrow$ hűséges $\surd$

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\Longleftrightarrow$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Példa: R(TNÉV, TERMELŐ, ÁR, CÍM)

$F = \{\text{TERMELŐ} \to \text{CÍM};\ \text{TNÉV, TERMELŐ} \to \text{ÁR}\}$

$\rho = (\text{TERMELŐ, CÍM} ; \text{TNÉV, TERMELŐ, ÁR})$

$R_1 \cap R_2 = \{\text{TERMELŐ}\}, \quad R_1 \setminus R_2 = \{\text{CÍM}\}, \quad R_2 \setminus R_1 = \{\text{TNÉV, ÁR}\}$

$\Longrightarrow (\text{TERMELŐ})^+(F) = \{\text{TERMELŐ, CÍM}\} \supseteq R_1 \setminus R_2 \Longrightarrow$ hűséges $\surd$

$\rho = (\text{TNÉV, TERMELŐ; TNÉV, CÍM, ÁR})$

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\Longleftrightarrow$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Példa: R(TNÉV, TERMELŐ, ÁR, CÍM)

$F = \{\text{TERMELŐ} \to \text{CÍM};\ \text{TNÉV, TERMELŐ} \to \text{ÁR}\}$

$\rho = (\text{TERMELŐ, CÍM} ; \text{TNÉV, TERMELŐ, ÁR})$

$R_1 \cap R_2 = \{\text{TERMELŐ}\}, \quad R_1 \setminus R_2 = \{\text{CÍM}\}, \quad R_2 \setminus R_1 = \{\text{TNÉV, ÁR}\}$

$\Longrightarrow (\text{TERMELŐ})^+(F) = \{\text{TERMELŐ, CÍM}\} \supseteq R_1 \setminus R_2 \Longrightarrow$ hűséges $\surd$

$\rho = (\text{TNÉV, TERMELŐ; TNÉV, CÍM, ÁR})$

$R_1 \cap R_2 = \{\text{TNÉV}\}, \quad R_1 \setminus R_2 = \{\text{TERMELŐ}\}, \quad R_2 \setminus R_1 = \{\text{CÍM, ÁR}\}$

Hűséges felbontás két részre

Hogyan biztosíthatja F, hogy a felbontás hűséges legyen?

Tétel

Az (R, F) séma $\rho = (R_1, R_2)$ felbontása hűséges $\iff$ vagy

(a) $F \models R_1 \cap R_2 \to R_1 \setminus R_2$, *vagy*

(b) $F \models R_1 \cap R_2 \to R_2 \setminus R_1$.

Példa: R(TNÉV, TERMELŐ, ÁR, CÍM)

$F = \{\text{TERMELŐ} \to \text{CÍM};\ \text{TNÉV, TERMELŐ} \to \text{ÁR}\}$

$\rho = (\text{TERMELŐ, CÍM} ; \text{TNÉV, TERMELŐ, ÁR})$

$R_1 \cap R_2 = \{\text{TERMELŐ}\}, \quad R_1 \setminus R_2 = \{\text{CÍM}\}, \quad R_2 \setminus R_1 = \{\text{TNÉV, ÁR}\}$

$\implies (\text{TERMELŐ})^+(F) = \{\text{TERMELŐ, CÍM}\} \supseteq R_1 \setminus R_2 \implies$ hűséges $\surd$

$\rho = (\text{TNÉV, TERMELŐ; TNÉV, CÍM, ÁR})$

$R_1 \cap R_2 = \{\text{TNÉV}\}, \quad R_1 \setminus R_2 = \{\text{TERMELŐ}\}, \quad R_2 \setminus R_1 = \{\text{CÍM, ÁR}\}$

$\implies (\text{TNÉV})^+(F)$ nem tartalmazza egyiket sem $\implies$ nem hűséges $\lightning$

Bizonyítás

Bizonyítás.

$\Longleftarrow$

greenTegyük fel, hogy $F \models R_1 \cap R_2 \rightarrow R_1 \setminus R_2$*, belátjuk, hogy a felbontás hűséges.*
(Ha a másik igaz, ugyanígy.)

Bizonyítás

Bizonyítás.

$\Longleftarrow$

greenTegyük fel, hogy $F \models R_1 \cap R_2 \to R_1 \setminus R_2$*, belátjuk, hogy a felbontás hűséges.*
(Ha a másik igaz, ugyanígy.)
Legyen r *egy tetszőleges reláció,* $s = m_\rho(r)$*. Elég belátni, hogy* $s \subseteq r$*, hiszen* $r \subseteq s$ *mindig igaz. Azaz, lássuk be, hogy ha* t *sora* s*-nek, akkor* r*-nek is.*

Bizonyítás

Bizonyítás.

$\Longleftarrow$

greenTegyük fel, hogy $F \models R_1 \cap R_2 \to R_1 \setminus R_2$*, belátjuk, hogy a felbontás hűséges.*
(Ha a másik igaz, ugyanígy.)
Legyen r *egy tetszőleges reláció,* $s = m_\rho(r)$*. Elég belátni, hogy* $s \subseteq r$*, hiszen* $r \subseteq s$ *mindig igaz. Azaz, lássuk be, hogy ha* t *sora* s*-nek, akkor* r*-nek is.*
Ha t *sora* s*-nek, akkor* $\exists u_1, u_2$ *sorai* r*-nek, hogy* $t[R_1] = u_1[R_1]$ *és* $t[R_2] = u_2[R_2]$*.*

Bizonyítás

Bizonyítás.

$\Longleftarrow$

greenTegyük fel, hogy $F \models R_1 \cap R_2 \to R_1 \setminus R_2$*, belátjuk, hogy a felbontás hűséges.*
(Ha a másik igaz, ugyanígy.)
Legyen r *egy tetszőleges reláció,* $s = m_\rho(r)$*. Elég belátni, hogy* $s \subseteq r$*, hiszen* $r \subseteq s$ *mindig igaz. Azaz, lássuk be, hogy ha* t *sora* s*-nek, akkor* r*-nek is.*
Ha t *sora* s*-nek, akkor* $\exists u_1, u_2$ *sorai* r*-nek, hogy* $t[R_1] = u_1[R_1]$ *és* $t[R_2] = u_2[R_2]$*.*
$\Longrightarrow u_1[R_1 \cap R_2] = t[R_1 \cap R_2] = u_2[R_1 \cap R_2]$

Bizonyítás

Bizonyítás.

$\Longleftarrow$

greenTegyük fel, hogy $F \models R_1 \cap R_2 \to R_1 \setminus R_2$*, belátjuk, hogy a felbontás hűséges.*
(Ha a másik igaz, ugyanígy.)
Legyen r *egy tetszőleges reláció,* $s = m_\rho(r)$*. Elég belátni, hogy* $s \subseteq r$*, hiszen* $r \subseteq s$ *mindig igaz. Azaz, lássuk be, hogy ha* t *sora* s*-nek, akkor* r*-nek is.*
Ha t *sora* s*-nek, akkor* $\exists u_1, u_2$ *sorai* r*-nek, hogy* $t[R_1] = u_1[R_1]$ *és* $t[R_2] = u_2[R_2]$*.*
$\Longrightarrow u_1[R_1 \cap R_2] = t[R_1 \cap R_2] = u_2[R_1 \cap R_2]$
de ha két sor megegyezik a metszeten, akkor a feltétel miatt $R_1 \setminus R_2$*-n is*
$\Longrightarrow$ *egyeznek az egész* R_1*-en* $\Longrightarrow u_2$ *és* t *egyeznek* R_1*-en.*

Bizonyítás

Bizonyítás.

$\Longleftarrow$

greenTegyük fel, hogy $F \models R_1 \cap R_2 \to R_1 \setminus R_2$*, belátjuk, hogy a felbontás hűséges.*
(Ha a másik igaz, ugyanígy.)
Legyen r *egy tetszőleges reláció,* $s = m_\rho(r)$*. Elég belátni, hogy* $s \subseteq r$*, hiszen* $r \subseteq s$ *mindig igaz. Azaz, lássuk be, hogy ha* t *sora* s*-nek, akkor* r*-nek is.*
Ha t *sora* s*-nek, akkor* $\exists u_1, u_2$ *sorai* r*-nek, hogy* $t[R_1] = u_1[R_1]$ *és* $t[R_2] = u_2[R_2]$*.*
$\Longrightarrow u_1[R_1 \cap R_2] = t[R_1 \cap R_2] = u_2[R_1 \cap R_2]$
de ha két sor megegyezik a metszeten, akkor a feltétel miatt $R_1 \setminus R_2$*-n is*
$\Longrightarrow$ *egyeznek az egész* R_1*-en* $\Longrightarrow u_2$ *és* t *egyeznek* R_1*-en.*
$\Longrightarrow t = u_2$*, hiszen* R_1*-en a fenti miatt,* R_2*-n a feltevés miatt egyeznek.* $\surd$

Bizonyítás

Bizonyítás.

$\Longleftarrow$

greenTegyük fel, hogy $F \models R_1 \cap R_2 \to R_1 \setminus R_2$*, belátjuk, hogy a felbontás hűséges. (Ha a másik igaz, ugyanígy.)*

Legyen r *egy tetszőleges reláció,* $s = m_\rho(r)$*. Elég belátni, hogy* $s \subseteq r$*, hiszen* $r \subseteq s$ *mindig igaz. Azaz, lássuk be, hogy ha* t *sora* s*-nek, akkor* r*-nek is.*

Ha t *sora* s*-nek, akkor* $\exists u_1, u_2$ *sorai* r*-nek, hogy* $t[R_1] = u_1[R_1]$ *és* $t[R_2] = u_2[R_2]$*.*

$\Longrightarrow u_1[R_1 \cap R_2] = t[R_1 \cap R_2] = u_2[R_1 \cap R_2]$

de ha két sor megegyezik a metszeten, akkor a feltétel miatt $R_1 \setminus R_2$*-n is*

$\Longrightarrow$ *egyeznek az egész* R_1*-en* $\Longrightarrow u_2$ *és* t *egyeznek* R_1*-en.*

$\Longrightarrow t = u_2$*, hiszen* R_1*-en a fenti miatt,* R_2*-n a feltevés miatt egyeznek.* $\surd$

$\Longrightarrow$ *Indirekt bizonyítunk. Tegyük fel, hogy sem (a), sem (b) nem igaz, azaz* $R_1 \setminus R_2 \not\subseteq (R_1 \cap R_2)^+(F)$ *és* $R_2 \setminus R_1 \not\subseteq (R_1 \cap R_2)^+(F)$*. Belátjuk, hogy ekkor* ρ *nem hűséges.*

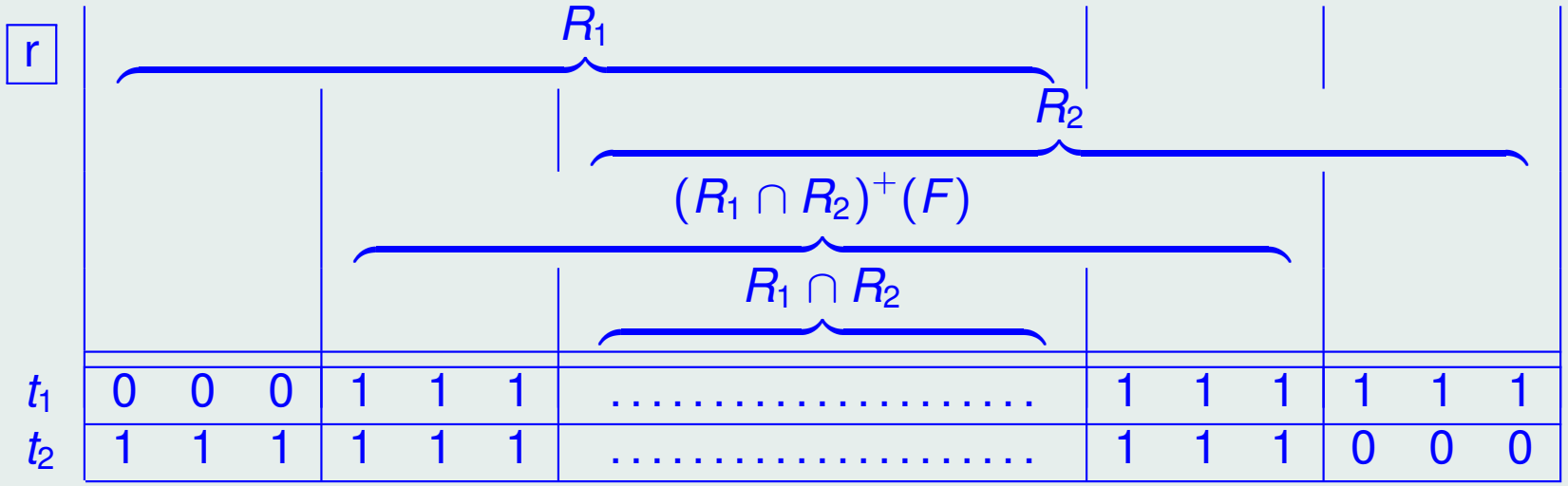
Bizonyítás.

Adunk egy olyan (R, F) sémát, ahol sem (a), sem (b) nem áll fenn, és megmutatjuk, hogy ezen a felbontás nem h?séges.

Bizonyítás.

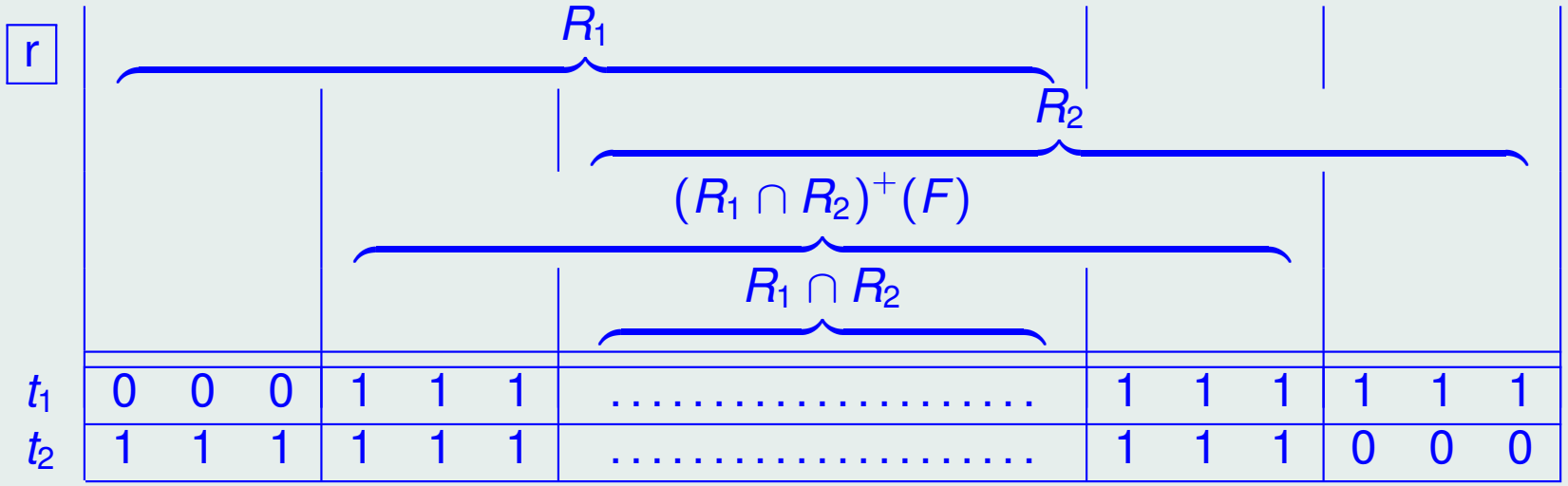
Adunk egy olyan (R, F) sémát, ahol sem (a), sem (b) nem áll fenn, és megmutatjuk, hogy ezen a felbontás nem h?séges.

Legyen r a következő reláció:



Bizonyítás.

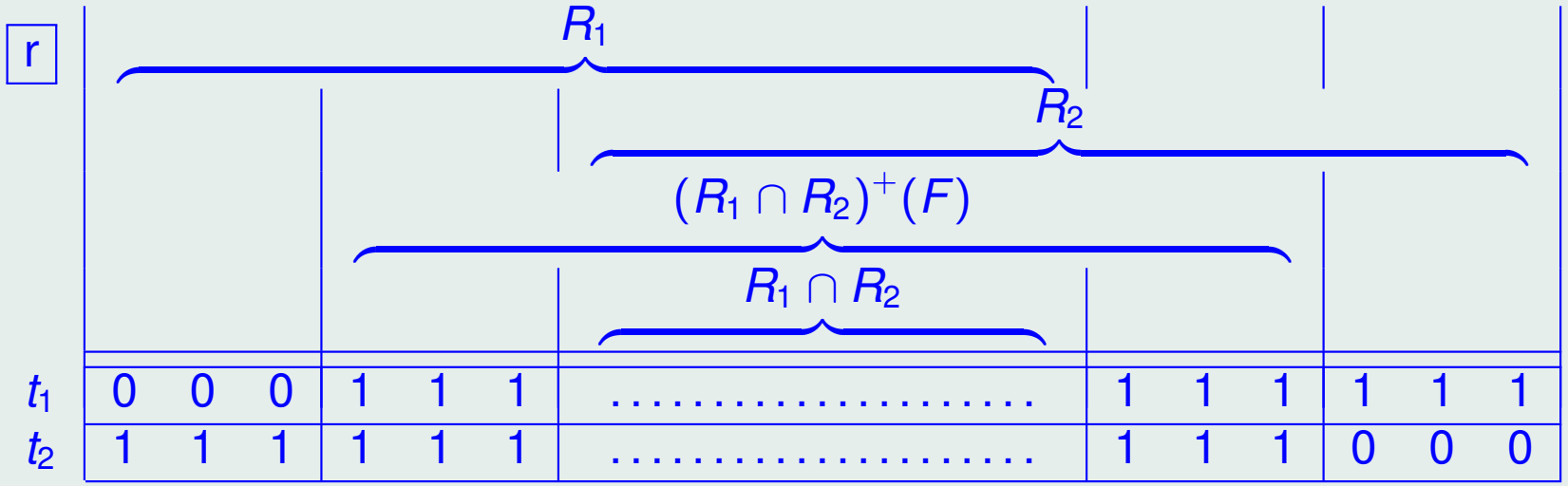
Adunk egy olyan (R, F) sémát, ahol sem (a), sem (b) nem áll fenn, és megmutatjuk, hogy ezen a felbontás nem h?séges.
Legyen r a következő reláció:



A feltétel miatt a két szélső rész nem üres, ott nem egyezik meg a két sor.

Bizonyítás.

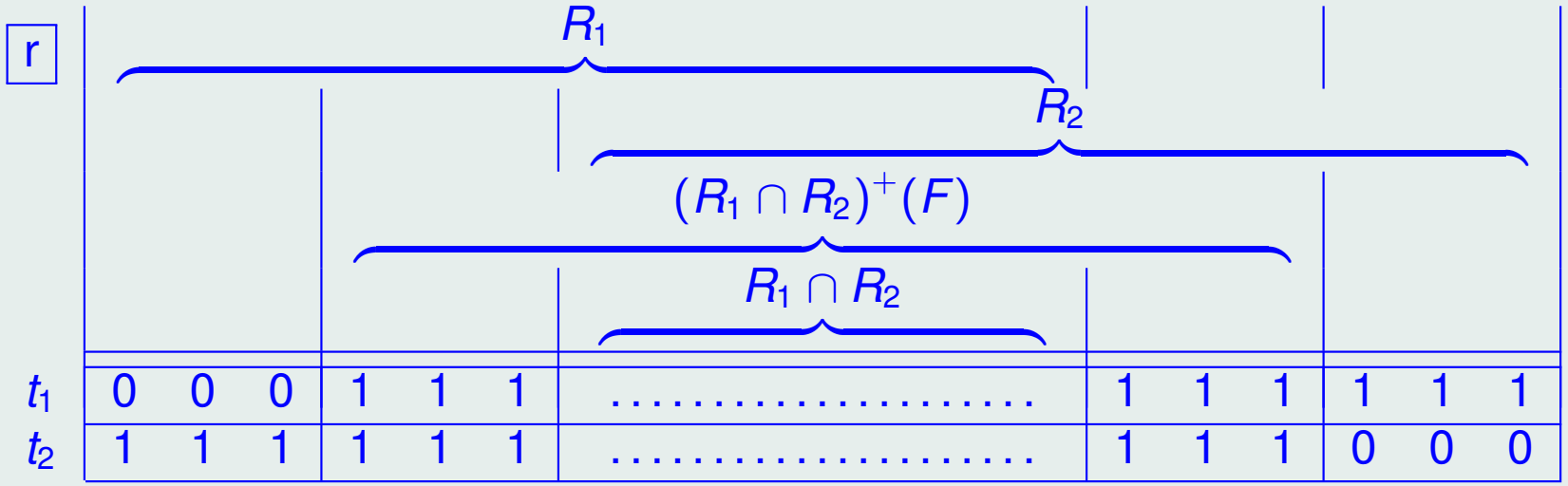
Adunk egy olyan (R, F) sémát, ahol sem (a), sem (b) nem áll fenn, és megmutatjuk, hogy ezen a felbontás nem h?séges.
Legyen r a következő reláció:



A feltétel miatt a két szélső rész nem üres, ott nem egyezik meg a két sor.
r-ben igazak F függőségei (mint a teljességi tételnél).

Bizonyítás.

Adunk egy olyan (R, F) sémát, ahol sem (a), sem (b) nem áll fenn, és megmutatjuk, hogy ezen a felbontás nem h?séges.
Legyen r a következő reláció:



A feltétel miatt a két szélső rész nem üres, ott nem egyezik meg a két sor.
r-ben igazak F függőségei (mint a teljességi tételnél).
Viszont $m_\rho(r) \supsetneq r$, hiszen $m_\rho(r)$-ben a csupa 1 sor is benne van. Tehát a felbontás nem h?séges.

□

Hűségesség ellenőrzése általában

Adott (R, F) és $\rho = (R_1, \ldots, R_k)$, ahol $R = A_1, \ldots, A_n$.

Hogyan tudjuk eldönteni, hogy hűséges-e a felbontás?

Hűségesség ellenőrzése általában

Adott (R, F) és $\rho = (R_1, \ldots, R_k)$, ahol $R = A_1, \ldots, A_n$.

Hogyan tudjuk eldönteni, hogy hűséges-e a felbontás?

Készítünk egy $k \times n$-es táblázatot:

	A_1	$\ldots$	A_j	$\ldots$	$A_{j'}$	$\ldots$	A_n
R_1							
$\vdots$							
R_i			a_j		$b_{ij'}$		
$\vdots$							
R_k							

- Kezdetben az (i, j) helyre a_j-t írunk, ha $A_j \in R_i$ és b_{ij}-t, ha $A_j \notin R_i$.

Hűségesség ellenőrzése általában

Adott (R, F) és $\rho = (R_1, \ldots, R_k)$, ahol $R = A_1, \ldots, A_n$.

Hogyan tudjuk eldönteni, hogy hűséges-e a felbontás?

Készítünk egy $k \times n$-es táblázatot:

	A_1	$\ldots$	A_j	$\ldots$	$A_{j'}$	$\ldots$	A_n
R_1							
$\vdots$							
R_i			a_j		$b_{ij'}$		
$\vdots$							
R_k							

- Kezdetben az (i, j) helyre a_j-t írunk, ha $A_j \in R_i$ és b_{ij}-t, ha $A_j \notin R_i$.
- Veszünk egy tetszőleges $X \to Y \in F$ függést.
  Ha két sor megegyezik X-en, akkor egyenlővé tesszük Y-on is az alábbi módon:

Hűségesség ellenőrzése általában

Adott (R, F) és $\rho = (R_1, \ldots, R_k)$, ahol $R = A_1, \ldots, A_n$.

Hogyan tudjuk eldönteni, hogy hűséges-e a felbontás?

Készítünk egy $k \times n$-es táblázatot:

	A_1	$\ldots$	A_j	$\ldots$	$A_{j'}$	$\ldots$	A_n
R_1							
$\vdots$							
R_i			a_j		$b_{ij'}$		
$\vdots$							
R_k							

- Kezdetben az (i, j) helyre a_j-t írunk, ha $A_j \in R_i$ és b_{ij}-t, ha $A_j \notin R_i$.
- Veszünk egy tetszőleges $X \to Y \in F$ függést.
  Ha két sor megegyezik X-en, akkor egyenlővé tesszük Y-on is az alábbi módon:
  - Ha valahol a_j és b_{ij} van, akkor a b_{ij}-t a_j-ra cseréljük.

Hűségesség ellenőrzése általában

Adott (R, F) és $\rho = (R_1, \ldots, R_k)$, ahol $R = A_1, \ldots, A_n$.

Hogyan tudjuk eldönteni, hogy hűséges-e a felbontás?

Készítünk egy $k \times n$-es táblázatot:

	A_1	$\ldots$	A_j	$\ldots$	$A_{j'}$	$\ldots$	A_n
R_1							
$\vdots$							
R_i			a_j		$b_{ij'}$		
$\vdots$							
R_k							

- Kezdetben az (i, j) helyre a_j-t írunk, ha $A_j \in R_i$ és b_{ij}-t, ha $A_j \notin R_i$.
- Veszünk egy tetszőleges $X \to Y \in F$ függést.
  Ha két sor megegyezik X-en, akkor egyenlővé tesszük Y-on is az alábbi módon:
  - Ha valahol a_j és b_{ij} van, akkor a b_{ij}-t a_j-ra cseréljük.
  - Ha b_{kj} és b_{lj} van, akkor az egyiket átírjuk a másikra.

Hűségesség ellenőrzése általában

Adott (R, F) és $\rho = (R_1, \ldots, R_k)$, ahol $R = A_1, \ldots, A_n$.

Hogyan tudjuk eldönteni, hogy hűséges-e a felbontás?

Készítünk egy $k \times n$-es táblázatot:

	A_1	$\ldots$	A_j	$\ldots$	$A_{j'}$	$\ldots$	A_n
R_1							
$\vdots$							
R_i			a_j		$b_{ij'}$		
$\vdots$							
R_k							

- Kezdetben az (i, j) helyre a_j-t írunk, ha $A_j \in R_i$ és b_{ij}-t, ha $A_j \notin R_i$.
- Veszünk egy tetszőleges $X \to Y \in F$ függést.
  Ha két sor megegyezik X-en, akkor egyenlővé tesszük Y-on is az alábbi módon:
  - Ha valahol a_j és b_{ij} van, akkor a b_{ij}-t a_j-ra cseréljük.
  - Ha b_{kj} és b_{lj} van, akkor az egyiket átírjuk a másikra.
- Ezt minden függésre megcsináljuk tetszőleges sorrendben, szükség esetén többször is.

Jó a módszer?

Tétel

ρ pontosan akkor hűséges ha a végén lesz csupa a sor.

Nem bizonyítjuk.

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1			

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	b_{13}	b_{14}

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	b_{13}	b_{14}
R_2				

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	b_{13}	b_{14}
R_2	b_{21}	a_2	a_3	b_{24}

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	b_{13}	b_{14}
R_2	b_{21}	a_2	a_3	b_{24}
R_3				

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	b_{13}	b_{14}
R_2	b_{21}	a_2	a_3	b_{24}
R_3	a_1	b_{32}	a_3	a_4

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	$b_{13} \to a_3^*$	b_{14}
R_2	b_{21}	a_2	a_3	b_{24}
R_3	a_1	b_{32}	a_3	a_4

* $A \to C$ miatt

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	$\to \boxed{a_3^*}$	b_{14}
R_2	b_{21}	a_2	a_3	b_{24}
R_3	a_1	$b_{32} \to \boxed{a_2^{**}}$	a_3	a_4

* $A \to C$ miatt
** $C \to B$ miatt

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	$\to \boxed{a_3^*}$	b_{14}
R_2	b_{21}	a_2	a_3	b_{24}
R_3	a_1	$\to \boxed{a_2^{**}}$	a_3	a_4

* $A \to C$ miatt
** $C \to B$ miatt

Lett csupa a sor $\Longrightarrow$ hűséges felbontás

Példa a táblázatos tesztre

$R(ABCD) \quad F = \{A \to C;\ C \to B;\ B \to D\}$
$R_1 = AB, R_2 = BC, R_3 = ACD \quad \rho = (R_1, R_2, R_3)$

	A	B	C	D
R_1	a_1	a_2	$b_{13} \to a_3^*$	b_{14}
R_2	b_{21}	a_2	a_3	b_{24}
R_3	a_1	$b_{32} \to a_2^{**}$	a_3	a_4

* $A \to C$ miatt
** $C \to B$ miatt

Lett csupa *a* sor $\Longrightarrow$ hűséges felbontás

Hűséges felbontás

Tétel

Adott (R, F), $\rho = (R_1, \ldots, R_k)$ az R hűséges felbontása és $\sigma = (S_1, \ldots, S_m)$ az R_1 hűséges felbontása (azaz R_1-et tovább bontjuk). Ekkor $\tau = (S_1, \ldots, S_m, R_2, \ldots, R_k)$ hűséges felbontása R-nek.

Hűséges felbontás

Tétel

Adott (R, F), $\rho = (R_1, \ldots, R_k)$ az R hűséges felbontása és $\sigma = (S_1, \ldots, S_m)$ az R_1 hűséges felbontása (azaz R_1-et tovább bontjuk). Ekkor $\tau = (S_1, \ldots, S_m, R_2, \ldots, R_k)$ hűséges felbontása R-nek.

Bizonyítás.

Legyen r egy R-re illeszkedő reláció és ennek R_1-re eső vetülete legyen r_1.

Hűséges felbontás

Tétel

Adott (R, F), $\rho = (R_1, \ldots, R_k)$ az R hűséges felbontása és $\sigma = (S_1, \ldots, S_m)$ az R_1 hűséges felbontása (azaz R_1-et tovább bontjuk). Ekkor $\tau = (S_1, \ldots, S_m, R_2, \ldots, R_k)$ hűséges felbontása R-nek.

Bizonyítás.

Legyen r egy R-re illeszkedő reláció és ennek R_1-re eső vetülete legyen r_1. Tovább bontva r_1-et σ szerint kapjuk az $s_1, s_2, \ldots, s_m$ vetületeket.

Hűséges felbontás

Tétel

Adott (R, F), $\rho = (R_1, \ldots, R_k)$ az R hűséges felbontása és $\sigma = (S_1, \ldots, S_m)$ az R_1 hűséges felbontása (azaz R_1-et tovább bontjuk). Ekkor $\tau = (S_1, \ldots, S_m, R_2, \ldots, R_k)$ hűséges felbontása R-nek.

Bizonyítás.

Legyen r egy R-re illeszkedő reláció és ennek R_1-re eső vetülete legyen r_1. Tovább bontva r_1-et σ szerint kapjuk az $s_1, s_2, \ldots, s_m$ vetületeket. Mivel σ hűséges, ezért $s_1 \bowtie s_2 \bowtie \cdots s_m = m_\sigma(r_1) = r_1$.

Hűséges felbontás

Tétel

Adott (R, F), $\rho = (R_1, \ldots, R_k)$ az R hűséges felbontása és $\sigma = (S_1, \ldots, S_m)$ az R_1 hűséges felbontása (azaz R_1-et tovább bontjuk). Ekkor $\tau = (S_1, \ldots, S_m, R_2, \ldots, R_k)$ hűséges felbontása R-nek.

Bizonyítás.

Legyen r egy R-re illeszkedő reláció és ennek R_1-re eső vetülete legyen r_1. Tovább bontva r_1-et σ szerint kapjuk az $s_1, s_2, \ldots, s_m$ vetületeket. Mivel σ hűséges, ezért $s_1 \bowtie s_2 \bowtie \cdots s_m = m_\sigma(r_1) = r_1$.
Mivel ρ is hűséges, ezért $r = m_\rho(r) = r_1 \bowtie r_2 \bowtie \cdots r_k$.

Hűséges felbontás

Tétel

Adott (R, F), $\rho = (R_1, \ldots, R_k)$ az R hűséges felbontása és $\sigma = (S_1, \ldots, S_m)$ az R_1 hűséges felbontása (azaz R_1-et tovább bontjuk). Ekkor $\tau = (S_1, \ldots, S_m, R_2, \ldots, R_k)$ hűséges felbontása R-nek.

Bizonyítás.

Legyen r egy R-re illeszkedő reláció és ennek R_1-re eső vetülete legyen r_1. Tovább bontva r_1-et σ szerint kapjuk az $s_1, s_2, \ldots, s_m$ vetületeket. Mivel σ hűséges, ezért $s_1 \bowtie s_2 \bowtie \cdots s_m = m_\sigma(r_1) = r_1$.
Mivel ρ is hűséges, ezért $r = m_\rho(r) = r_1 \bowtie r_2 \bowtie \cdots r_k$. Ebbe beírva r_1 helyére a σ hűségességéből kapott egyenlőséget, kapjuk, hogy
$r = s_1 \bowtie s_2 \bowtie \cdots s_m \bowtie r_2 \bowtie \cdots r_k = m_\tau(r)$, azaz τ is hűséges. Itt persze használtuk $\bowtie$ asszociativitását.

Hűséges felbontás

Tétel

Adott (R, F), $\rho = (R_1, \ldots, R_k)$ az R hűséges felbontása és $\sigma = (S_1, \ldots, S_m)$ az R_1 hűséges felbontása (azaz R_1-et tovább bontjuk). Ekkor $\tau = (S_1, \ldots, S_m, R_2, \ldots, R_k)$ hűséges felbontása R-nek.

Bizonyítás.

Legyen r egy R-re illeszkedő reláció és ennek R_1-re eső vetülete legyen r_1. Tovább bontva r_1-et σ szerint kapjuk az $s_1, s_2, \ldots, s_m$ vetületeket. Mivel σ hűséges, ezért $s_1 \bowtie s_2 \bowtie \cdots s_m = m_\sigma(r_1) = r_1$.
Mivel ρ is hűséges, ezért $r = m_\rho(r) = r_1 \bowtie r_2 \bowtie \cdots r_k$. Ebbe beírva r_1 helyére a σ hűségességéből kapott egyenlőséget, kapjuk, hogy
$r = s_1 \bowtie s_2 \bowtie \cdots s_m \bowtie r_2 \bowtie \cdots r_k = m_\tau(r)$, azaz τ is hűséges. Itt persze használtuk $\bowtie$ asszociativitását. □

Hűséges felbontás

Definíció

Ha egy σ felbontást a ρ felbontásból, úgy kapjuk, hogy néhány relációját felbontjuk, akkor ezt a következőképp jelöljük: $\sigma \supseteq \rho$.

Tétel

Ha ρ hűséges és $\sigma \supseteq \rho$ (σ-ban több komponens van), akkor σ is hűséges.

Hűséges felbontás

Definíció

Ha egy σ felbontást a ρ felbontásból, úgy kapjuk, hogy néhány relációját felbontjuk, akkor ezt a következőképp jelöljük: $\sigma \supseteq \rho$.

Tétel

Ha ρ hűséges és $\sigma \supseteq \rho$ (σ-ban több komponens van), akkor σ is hűséges.

Bizonyítás.

$r \subseteq m_\sigma(r) \subseteq m_\rho(r) = r$

Hűséges felbontás

Definíció

Ha egy σ felbontást a ρ felbontásból, úgy kapjuk, hogy néhány relációját felbontjuk, akkor ezt a következőképp jelöljük: $\sigma \supseteq \rho$.

Tétel

Ha ρ hűséges és $\sigma \supseteq \rho$ (σ-ban több komponens van), akkor σ is hűséges.

Bizonyítás.

$r \subseteq m_\sigma(r) \subseteq m_\rho(r) = r$
A középső tartalmazás azért igaz, mert a keresztszorzatból szigorúbb feltételek szerint válogatunk.

Hűséges felbontás

Definíció

Ha egy σ felbontást a ρ felbontásból, úgy kapjuk, hogy néhány relációját felbontjuk, akkor ezt a következőképp jelöljük: $\sigma \supseteq \rho$.

Tétel

Ha ρ hűséges és $\sigma \supseteq \rho$ (σ-ban több komponens van), akkor σ is hűséges.

Bizonyítás.

$r \subseteq m_\sigma(r) \subseteq m_\rho(r) = r$
A középső tartalmazás azért igaz, mert a keresztszorzatból szigorúbb feltételek szerint válogatunk.
$\Longrightarrow r = m_\sigma(r) \quad \surd$ □

Adatbázisok elmélete

Normálformák, BCNF

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

14. előadás

Normálformák

Definíció

Egy $X \to Y$ függés triviális, ha $Y \subseteq X$. (Mert ezek a függések nem hordoznak sok infót, mindig igazak.)

Normálformák

Definíció

Egy $X \to Y$ függés triviális, ha $Y \subseteq X$. (Mert ezek a függések nem hordoznak sok infót, mindig igazak.)

Definíció (Boyce–Codd normálforma)

Az (R, F) relációs séma BCNF-ben van, ha tetszőleges nemtriviális $X \to A \in F^+$ függés esetén X szuperkulcs.

Normálformák

Definíció

Egy $X \to Y$ függés triviális, ha $Y \subseteq X$. (Mert ezek a függések nem hordoznak sok infót, mindig igazak.)

Definíció (Boyce–Codd normálforma)

Az (R, F) relációs séma BCNF-ben van, ha tetszőleges nemtriviális $X \to A \in F^+$ függés esetén X szuperkulcs.

Azaz csak olyan függések vannak, hogy a szuperkulcs mindent meghatároz.

Normálformák

Tétel

Az (R, F) BCNF-ben van pontosan akkor ha tetszőleges $A \in R$-re és $X \subseteq R$ kulcsra igaz, hogy nincs olyan $Y \subseteq R$, amire $X \to Y \in F^+$; $Y \not\to X$; $Y \to A \in F^+$ és $A \notin Y$.

Normálformák

Tétel

Az (R, F) BCNF-ben van pontosan akkor ha tetszőleges $A \in R$-re és $X \subseteq R$ kulcsra igaz, hogy nincs olyan $Y \subseteq R$, amire $X \to Y \in F^+$; $Y \not\to X$; $Y \to A \in F^+$ és $A \notin Y$. (Nincs tranzitív függés kulcstól.)

Tétel

Az (R, F) BCNF-ben van pontosan akkor ha tetszőleges $A \in R$-re és $X \subseteq R$ kulcsra igaz, hogy nincs olyan $Y \subseteq R$, amire $X \to Y \in F^+$; $Y \not\to X$; $Y \to A \in F^+$ és $A \notin Y$. (Nincs tranzitív függés kulcstól.)

Bizonyítás.

Ha nincs BCNF-ben a séma, akkor van egy $Y \to A$ függés, ahol Y nem szuperkulcs és $A \notin Y$. Ekkor, tetszőleges X kulccsal: $X \to Y$, $Y \not\to X$, $Y \to A$, de $A \notin Y$, ami épp egy kulcstól való tranzitív függés.

Normálformák

Tétel

Az (R, F) BCNF-ben van pontosan akkor ha tetszőleges $A \in R$-re és $X \subseteq R$ kulcsra igaz, hogy nincs olyan $Y \subseteq R$, amire $X \to Y \in F^+$; $Y \not\to X$; $Y \to A \in F^+$ és $A \notin Y$. (Nincs tranzitív függés kulcstól.)

Bizonyítás.

Ha nincs BCNF-ben a séma, akkor van egy $Y \to A$ függés, ahol Y nem szuperkulcs és $A \notin Y$. Ekkor, tetszőleges X kulccsal: $X \to Y$, $Y \not\to X$, $Y \to A$, de $A \notin Y$, ami épp egy kulcstól való tranzitív függés.
Másrészt, ha van tranzitív függés kulcstól, azaz X olyan kulcs, amivel $X \to Y$, $Y \not\to X$, $Y \to A$, de $A \notin Y$, akkor $Y \to A$ egy olyan függés, ami sérti a BCNF tulajdonságot, mert Y nem lehet szuperkulcs, ha $Y \not\to X$. $\surd$ $\square$

Miért jó a BCNF séma?

Ha $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk

Miért jó a BCNF séma?

Ha $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk $\Longrightarrow$ **redundancia**.

Miért jó a BCNF séma?

Ha $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk $\Longrightarrow$ **redundancia**.

Állítás

$\leq$ 2 attribútumos reláció mindig BCNF.

Miért jó a BCNF séma?

Ha $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk $\Longrightarrow$ **redundancia**.

Állítás

$\leq$ 2 attribútumos reláció mindig BCNF.

Bizonyítás.

Ha $A \to B \implies A$ kulcs.

Miért jó a BCNF séma?

Ha $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk $\Longrightarrow$ **redundancia**.

Állítás

$\leq$ 2 attribútumos reláció mindig BCNF.

Bizonyítás.

Ha $A \to B \implies A$ kulcs. Ha $B \to A \implies B$ kulcs.

Miért jó a BCNF séma?

Ha $C \to B;\ B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk $\Longrightarrow$ **redundancia**.

Állítás

$\leq$ 2 attribútumos reláció mindig BCNF.

Bizonyítás.

Ha $A \to B \implies A$ kulcs. Ha $B \to A \implies B$ kulcs. □

Hogyan döntsük el, hogy egy (R, F) séma BCNF-e?

Miért jó a BCNF séma?

Ha $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk $\Longrightarrow$ **redundancia**.

Állítás

$\leq$ 2 attribútumos reláció mindig BCNF.

Bizonyítás.

Ha $A \to B \implies A$ kulcs. Ha $B \to A \implies B$ kulcs. □

Hogyan döntsük el, hogy egy (R, F) séma BCNF-e?
F^+ összes függ?ségét végig kellene nézni.

Miért jó a BCNF séma?

Ha $C \to B$; $B \to A$ teljesülne, de $B \to C$ nem, akkor ugyanaz a B érték több C érték mellett is előfordulhatna, de minden példánynál ugyanazt az A értéket is tároljuk $\Longrightarrow$ **redundancia**.

Állítás

$\leq$ 2 attribútumos reláció mindig BCNF.

Bizonyítás.

Ha $A \to B \implies A$ kulcs. Ha $B \to A \implies B$ kulcs. □

Hogyan döntsük el, hogy egy (R, F) séma BCNF-e?
F^+ összes függ?ségét végig kellene nézni.
DE:

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F)$

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \quad \Longrightarrow U \subsetneq U^+(F)$

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \quad \Longrightarrow U \subsetneq U^+(F)$
Az algoritmus, ami $U^+(F)$-et számolja, el tud indulni

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \quad \Longrightarrow U \subsetneq U^+(F)$
Az algoritmus, ami $U^+(F)$-et számolja, el tud indulni $\Longrightarrow \exists V \to W \in F$, melyre $V \subseteq U,\ W \nsubseteq U$

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \quad \Longrightarrow U \subsetneq U^+(F)$
Az algoritmus, ami $U^+(F)$-et számolja, el tud indulni $\Longrightarrow \exists V \to W \in F$, melyre $V \subseteq U,\ W \nsubseteq U \quad \Longrightarrow V \to W$ jó lesz $X \to Y$-nak.

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \quad \Longrightarrow U \subsetneq U^+(F)$
Az algoritmus, ami $U^+(F)$-et számolja, el tud indulni $\quad \Longrightarrow \exists V \to W \in F$, melyre $V \subseteq U,\ W \not\subseteq U \quad \Longrightarrow V \to W$ jó lesz $X \to Y$-nak.
Ugyanis V nem szuperkulcs, hiszen $V \subseteq U$ és U nem szuperkulcs.

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \quad \Longrightarrow U \subsetneq U^+(F)$
Az algoritmus, ami $U^+(F)$-et számolja, el tud indulni $\quad \Longrightarrow \exists V \to W \in F$, melyre $V \subseteq U$, $W \not\subseteq U \quad \Longrightarrow V \to W$ jó lesz $X \to Y$-nak.
Ugyanis V nem szuperkulcs, hiszen $V \subseteq U$ és U nem szuperkulcs.
$W \not\subseteq U$

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \quad \Longrightarrow U \subsetneq U^+(F)$
Az algoritmus, ami $U^+(F)$-et számolja, el tud indulni $\Longrightarrow \exists V \to W \in F$, melyre $V \subseteq U$, $W \not\subseteq U \quad \Longrightarrow V \to W$ jó lesz $X \to Y$-nak.
Ugyanis V nem szuperkulcs, hiszen $V \subseteq U$ és U nem szuperkulcs.
$W \not\subseteq U \quad \Longrightarrow \exists A \in W \setminus U \subseteq W \setminus V$, így $V \to W$ nem triviális.

Miért jó a BCNF séma?

Tétel

Ha (R, F) nem BCNF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális és X nem szuperkulcs. (Az ilyen $X \to A \in F^+$.)

Bizonyítás.

Ha (R, F) nem BCNF, akkor van $U \to B \in F^+$, hogy U nem szuperkulcs és $B \notin U$.
$\Longrightarrow B \in U^+(F) \Longrightarrow U \subsetneq U^+(F)$
Az algoritmus, ami $U^+(F)$-et számolja, el tud indulni $\Longrightarrow \exists V \to W \in F$, melyre $V \subseteq U$, $W \not\subseteq U \Longrightarrow V \to W$ jó lesz $X \to Y$-nak.
Ugyanis V nem szuperkulcs, hiszen $V \subseteq U$ és U nem szuperkulcs.
$W \not\subseteq U \Longrightarrow \exists A \in W \setminus U \subseteq W \setminus V$, így $V \to W$ nem triviális. □

Ez jelentősen könnyíti az ellenőrzést, csak F függőségeit kell végignézni, nem F^+-ét.

Normalizálás

Tétel

Tetszőleges (R, F) sémának van hűséges felbontása BCNF relációkra.

Normalizálás

Tétel

Tetszőleges (R, F) sémának van hűséges felbontása BCNF relációkra.

Bizonyítás.

Elve:

- *Ha (R, F) BCNF* $\surd$

Normalizálás

Tétel

Tetszőleges (R, F) sémának van hűséges felbontása BCNF relációkra.

Bizonyítás.

Elve:

- *Ha (R, F) BCNF* $\surd$
- *Ha nem, akkor két valódi (kisebb) részre bontjuk hűségesen* $\Longrightarrow (R_1, R_2)$

Normalizálás

Tétel

Tetszőleges (R, F) sémának van hűséges felbontása BCNF relációkra.

Bizonyítás.

Elve:

- *Ha (R, F) BCNF* $\surd$
- *Ha nem, akkor két valódi (kisebb) részre bontjuk hűségesen* $\Longrightarrow (R_1, R_2)$
- *Ezt ismételjük (R_1, R_2)-re.*

Normalizálás

Tétel

Tetszőleges (R, F) sémának van hűséges felbontása BCNF relációkra.

Bizonyítás.

Elve:

- *Ha (R, F) BCNF* $\surd$
- *Ha nem, akkor két valódi (kisebb) részre bontjuk hűségesen* $\Longrightarrow (R_1, R_2)$
- *Ezt ismételjük (R_1, R_2)-re.*

Ez véget fog érni, mert ha már csak 2 attribútum marad valamelyikben, azt nem kell tovább bontani.

Normalizálás

Tétel

Tetszőleges (R, F) sémának van hűséges felbontása BCNF relációkra.

Bizonyítás.

Elve:

- *Ha (R, F) BCNF* $\surd$
- *Ha nem, akkor két valódi (kisebb) részre bontjuk hűségesen* $\Longrightarrow (R_1, R_2)$
- *Ezt ismételjük (R_1, R_2)-re.*

Ez véget fog érni, mert ha már csak 2 attribútum marad valamelyikben, azt nem kell tovább bontani.
Hűséges lesz, mert láttuk, hogy ha egy hűséges felbontás egyik részét tovább bontjuk, akkor hűséges marad.

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot $\Longrightarrow$ A és X része a sémának, $A \notin X$ és X nem szuperkulcs

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot $\Longrightarrow$ A és X része a sémának, $A \notin X$ és X nem szuperkulcs

$$R_1 := XA, \qquad R_2 := R \setminus \{A\}$$

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot $\Longrightarrow$ A és X része a sémának, $A \notin X$ és X nem szuperkulcs

$$R_1 := XA, \qquad R_2 := R \setminus \{A\}$$

Ezek kisebbek: R_2 nyilván, R_1 pedig azért, mert ha $R_1 = R$ volna, akkor $X \to XA = R$ miatt X szuperkulcs lett volna.

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot $\Longrightarrow$ A és X része a sémának, $A \notin X$ és X nem szuperkulcs

$$R_1 := XA, \qquad R_2 := R \setminus \{A\}$$

Ezek kisebbek: R_2 nyilván, R_1 pedig azért, mert ha $R_1 = R$ volna, akkor $X \to XA = R$ miatt X szuperkulcs lett volna.

Hűséges a felbontás: kétrészes teszttel $R_1 \cap R_2 = X \to A = R_1 \setminus R_2$ $\surd$

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot $\Longrightarrow$ A és X része a sémának, $A \notin X$ és X nem szuperkulcs

$$R_1 := XA, \qquad R_2 := R \setminus \{A\}$$

Ezek kisebbek: R_2 nyilván, R_1 pedig azért, mert ha $R_1 = R$ volna, akkor $X \to XA = R$ miatt X szuperkulcs lett volna.

Hűséges a felbontás: kétrészes teszttel $R_1 \cap R_2 = X \to A = R_1 \setminus R_2$ $\surd$

□

Miért lesz jobb ez a felbontás?

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot $\Longrightarrow$ A és X része a sémának, $A \notin X$ és X nem szuperkulcs

$$R_1 := XA, \qquad R_2 := R \setminus \{A\}$$

Ezek kisebbek: R_2 nyilván, R_1 pedig azért, mert ha $R_1 = R$ volna, akkor $X \to XA = R$ miatt X szuperkulcs lett volna.

Hűséges a felbontás: kétrészes teszttel $R_1 \cap R_2 = X \to A = R_1 \setminus R_2$ $\surd$

□

Miért lesz jobb ez a felbontás?

Az $X \to A$ függéssel nem lesz több probléma:

Bizonyítás

Bizonyítás.

Hogyan bontjuk fel 2 valódi részre, hűségesen?

Keresünk a felbontandó sémában egy olyan $X \to A \in F^+$-t, ami sérti a BCNF tulajdonságot $\Longrightarrow$ A és X része a sémának, $A \notin X$ és X nem szuperkulcs

$$R_1 := XA, \qquad R_2 := R \setminus \{A\}$$

Ezek kisebbek: R_2 nyilván, R_1 pedig azért, mert ha $R_1 = R$ volna, akkor $X \to XA = R$ miatt X szuperkulcs lett volna.

Hűséges a felbontás: kétrészes teszttel $R_1 \cap R_2 = X \to A = R_1 \setminus R_2$ $\surd$

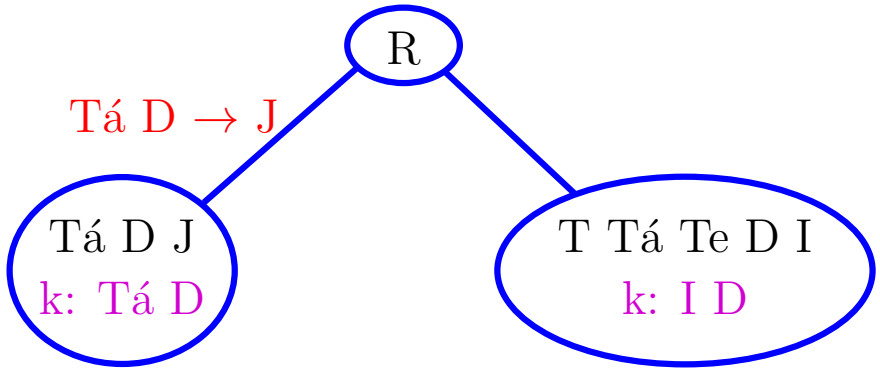
□

Miért lesz jobb ez a felbontás?

Az $X \to A$ függéssel nem lesz több probléma: R_2-ben nincs A, így nem lehet baj. R_1-ben viszont X szuperkulcs lesz.

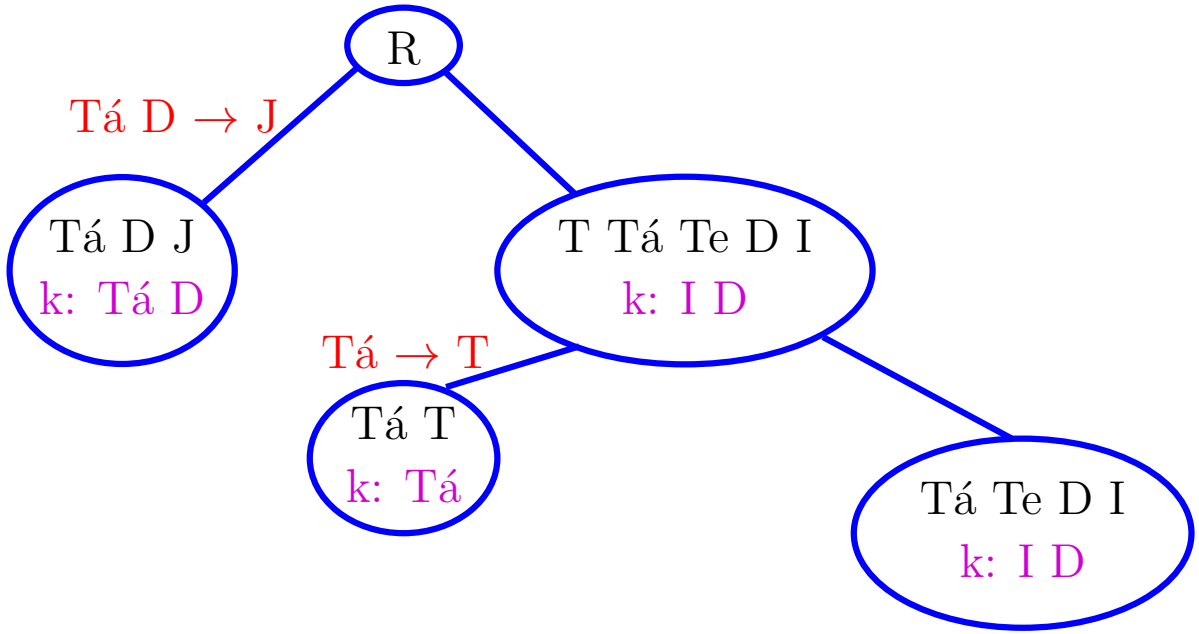
Példa

R(**T**anár, **Tá**rgy, **Te**rem, **D**iák, **J**egy, **I**dő)
$F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\} \implies$ kulcs csak ID



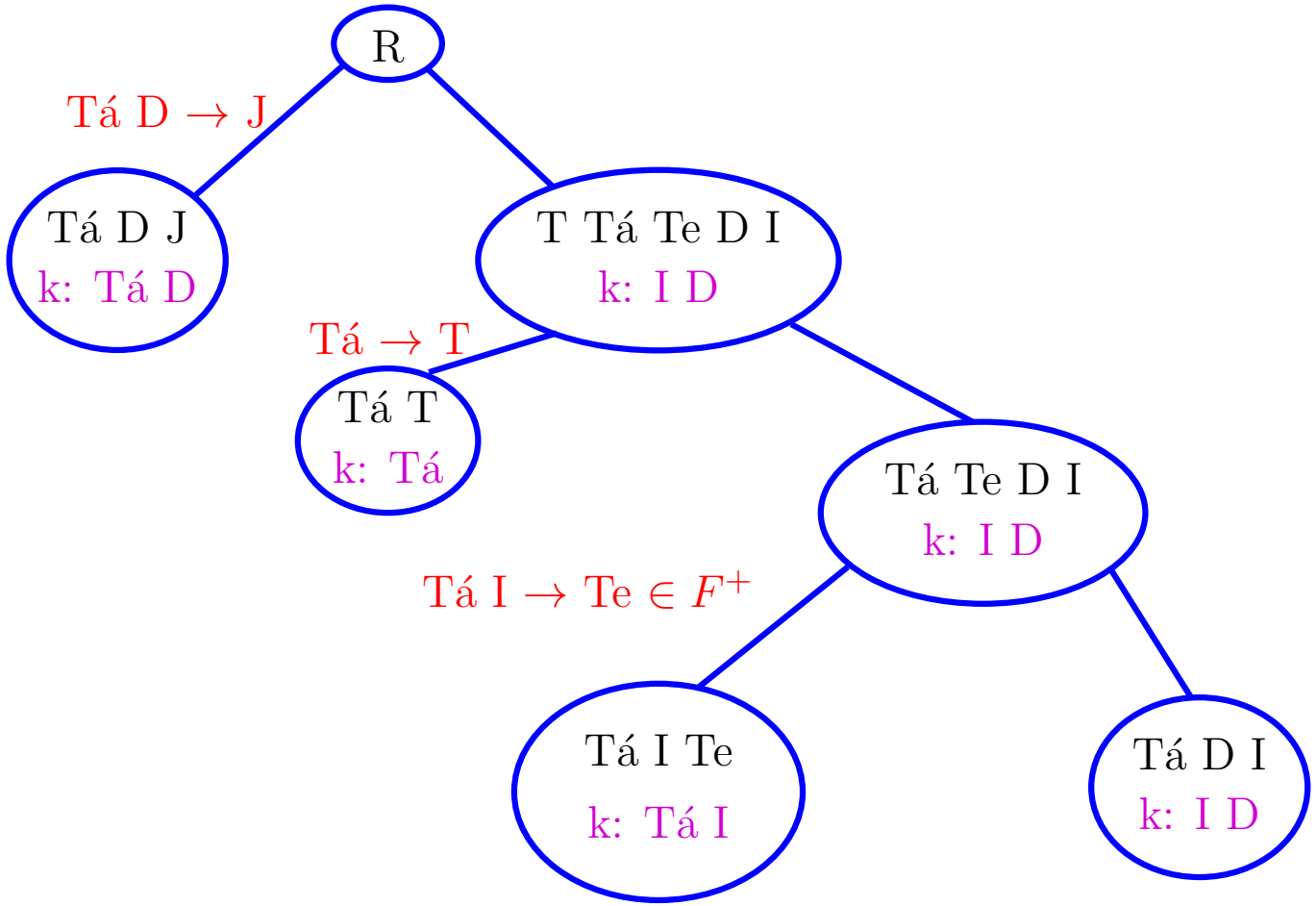
Példa

R(**T**anár, **Tá**rgy, **Te**rem, **D**iák, **J**egy, **I**dő)
$F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\} \implies$ kulcs csak ID



Példa

R(**T**anár, **Tá**rgy, **Te**rem, **D**iák, **J**egy, **I**dő)
$F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\} \implies$ kulcs csak ID



Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e.

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e. Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van.

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e. Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e. Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.

Általában nem igaz, hogy elég F-ből kiválogatni azokat, amiknek mindkét oldala S-ben van.

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e.
Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.
Általában nem igaz, hogy elég F-ből kiválogatni azokat, amiknek mindkét oldala S-ben van.
Pl.: $F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\}$
Ha $S = \text{Tá Te D I}$, akkor (csak a nemtrivi függéseket felírva):
$F_S^+ = \{\text{Tá I} \to \text{Te};\ \text{D I} \to \text{Te Tá};\ \text{D I Tá} \to \text{Te};\ \text{D I Te} \to \text{Tá}\}$

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e.
Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.
Általában nem igaz, hogy elég F-ből kiválogatni azokat, amiknek mindkét oldala S-ben van.
Pl.: $F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\}$
Ha $S = \text{Tá Te D I}$, akkor (csak a nemtrivi függéseket felírva):
$F_S^+ = \{\text{Tá I} \to \text{Te};\ \text{D I} \to \text{Te Tá};\ \text{D I Tá} \to \text{Te};\ \text{D I Te} \to \text{Tá}\}$
$\Longrightarrow$ Az előző algoritmus lehet exponenciális

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e.
Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.
Általában nem igaz, hogy elég F-ből kiválogatni azokat, amiknek mindkét oldala S-ben van.
Pl.: $F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\}$
Ha $S = \text{Tá Te D I}$, akkor (csak a nemtrivi függéseket felírva):
$F_S^+ = \{\text{Tá I} \to \text{Te};\ \text{D I} \to \text{Te Tá};\ \text{D I Tá} \to \text{Te};\ \text{D I Te} \to \text{Tá}\}$
$\Longrightarrow$ Az előző algoritmus lehet exponenciális $\Longrightarrow$ Van polinomiális algoritmus is.

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e.
Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.
Általában nem igaz, hogy elég F-ből kiválogatni azokat, amiknek mindkét oldala S-ben van.
Pl.: $F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\}$
Ha $S = \text{Tá Te D I}$, akkor (csak a nemtrivi függéseket felírva):
$F_S^+ = \{\text{Tá I} \to \text{Te};\ \text{D I} \to \text{Te Tá};\ \text{D I Tá} \to \text{Te};\ \text{D I Te} \to \text{Tá}\}$
$\Longrightarrow$ Az előző algoritmus lehet exponenciális $\Longrightarrow$ Van polinomiális algoritmus is.
3 attribútum esetén a BCNF tulajdonság csak úgy sérülhet, ha $X \to Y$, ahol X, Y egy-egy attribútum és X nem kulcs.

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e.
Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.
Általában nem igaz, hogy elég F-ből kiválogatni azokat, amiknek mindkét oldala S-ben van.
Pl.: $F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\}$
Ha $S = \text{Tá Te D I}$, akkor (csak a nemtrivi függéseket felírva):
$F_S^+ = \{\text{Tá I} \to \text{Te};\ \text{D I} \to \text{Te Tá};\ \text{D I Tá} \to \text{Te};\ \text{D I Te} \to \text{Tá}\}$
$\Longrightarrow$ Az előző algoritmus lehet exponenciális $\Longrightarrow$ Van polinomiális algoritmus is.
3 attribútum esetén a BCNF tulajdonság csak úgy sérülhet, ha $X \to Y$, ahol X, Y egy-egy attribútum és X nem kulcs.
Azt is mindig ellenőrizni kell, hogy a kapott relációkban mik a (szuper)kulcsok, hogy egy függésről el tudjuk dönteni, hogy sérti-e a BCNF-et vagy nem.

Megjegyzések

Minden felbontás után meg kell nézni, hogy a kapott relációk BCNF-ben vannak-e. Ehhez meg kell konstruálni F_S^+-et, ha S a vizsgált reláció: ez az F_R^+ azon függéseiből áll, amiknek mindkét oldala S-ben van. Ezeket a függéseket úgy kapjuk, hogy minden $X \subseteq S$ részhalmazra kiszámoljuk $X^+(F)$-et és $X \to Y$ pontosan akkor lesz benne F_S^+-ben, ha $Y \subseteq X^+(F) \cap S$.

Általában nem igaz, hogy elég F-ből kiválogatni azokat, amiknek mindkét oldala S-ben van.

Pl.: $F = \{\text{Tá} \to \text{T};\ \text{IT} \to \text{Te};\ \text{ID} \to \text{Te};\ \text{ID} \to \text{Tá};\ \text{TáD} \to \text{J}\}$

Ha $S = \text{Tá Te D I}$, akkor (csak a nemtrivi függéseket felírva):

$F_S^+ = \{\text{Tá I} \to \text{Te};\ \text{D I} \to \text{Te Tá};\ \text{D I Tá} \to \text{Te};\ \text{D I Te} \to \text{Tá}\}$

$\Longrightarrow$ Az előző algoritmus lehet exponenciális $\Longrightarrow$ Van polinomiális algoritmus is.

3 attribútum esetén a BCNF tulajdonság csak úgy sérülhet, ha $X \to Y$, ahol X, Y egy-egy attribútum és X nem kulcs.

Azt is mindig ellenőrizni kell, hogy a kapott relációkban mik a (szuper)kulcsok, hogy egy függésről el tudjuk dönteni, hogy sérti-e a BCNF-et vagy nem. A példában ez viszonylag könnyű lesz, hiszen I és D egyik F-beli függőségben sem szerepel a jobb oldalon, így minden kulcs (amikor I és D szerepel a relációban) tartalmazza I D-t. Csak azt kell megnézni, hogy I D kulcs marad.

Függőség megőrzése

BCNF egy fogyatékossága: nehéz lehet ellenőrizni, hogy teljesülnek-e F függései (pl. beszúráskor). Ilyenkor a költséges $\bowtie$ kell, és ez sokszor előfordulhat.

Függőség megőrzése

BCNF egy fogyatékossága: nehéz lehet ellenőrizni, hogy teljesülnek-e F függései (pl. beszúráskor). Ilyenkor a költséges $\bowtie$ kell, és ez sokszor előfordulhat.
Kéne egy olyan felbontás, amin könnyen lehet ellenőrizni a függéseket.

Függőség megőrzése

BCNF egy fogyatékossága: nehéz lehet ellenőrizni, hogy teljesülnek-e F függései (pl. beszúráskor). Ilyenkor a költséges $\bowtie$ kell, és ez sokszor előfordulhat.
Kéne egy olyan felbontás, amin könnyen lehet ellenőrizni a függéseket.

Definíció

Adott (R, F) séma és ennek egy $\rho = (R_1, \ldots, R_k)$ felbontása.

$$\pi_\rho(F) := \{X \to Y \in F^+ \mid \exists i\ (1 \leq i \leq k)\ X, Y \subseteq R_i\}^+$$

az F ***függéseinek vetítése a ρ felbontásra.***

Függőség megőrzése

BCNF egy fogyatékossága: nehéz lehet ellenőrizni, hogy teljesülnek-e F függései (pl. beszúráskor). Ilyenkor a költséges $\bowtie$ kell, és ez sokszor előfordulhat.
Kéne egy olyan felbontás, amin könnyen lehet ellenőrizni a függéseket.

Definíció

Adott (R, F) séma és ennek egy $\rho = (R_1, \ldots, R_k)$ felbontása.

$$\pi_\rho(F) := \{X \to Y \in F^+ \mid \exists i\ (1 \leq i \leq k)\ X, Y \subseteq R_i\}^+$$

*az F **függéseinek vetítése a ρ felbontásra**. ρ **függőségőrző**, ha $\pi_\rho(F) = F^+$.*

Függőség megőrzése

BCNF egy fogyatékossága: nehéz lehet ellenőrizni, hogy teljesülnek-e F függései (pl. beszúráskor). Ilyenkor a költséges $\bowtie$ kell, és ez sokszor előfordulhat.
Kéne egy olyan felbontás, amin könnyen lehet ellenőrizni a függéseket.

Definíció

Adott (R, F) séma és ennek egy $\rho = (R_1, \ldots, R_k)$ felbontása.

$$\pi_\rho(F) := \{X \to Y \in F^+ \mid \exists i\ (1 \leq i \leq k)\ X, Y \subseteq R_i\}^+$$

*az F **függéseinek vetítése a ρ felbontásra**. ρ **függőségőrző**, ha $\pi_\rho(F) = F^+$.*

Megjegyzés: $\pi_\rho(F) \subseteq F^+$ persze mindig igaz.

Függőség megőrzése

BCNF egy fogyatékossága: nehéz lehet ellenőrizni, hogy teljesülnek-e F függései (pl. beszúráskor). Ilyenkor a költséges $\bowtie$ kell, és ez sokszor előfordulhat.
Kéne egy olyan felbontás, amin könnyen lehet ellenőrizni a függéseket.

Definíció

Adott (R, F) séma és ennek egy $\rho = (R_1, \ldots, R_k)$ felbontása.

$$\pi_\rho(F) := \{X \to Y \in F^+ \mid \exists i\ (1 \leq i \leq k)\ X, Y \subseteq R_i\}^+$$

*az F **függéseinek vetítése a ρ felbontásra**. ρ **függőségőrző**, ha $\pi_\rho(F) = F^+$.*

Megjegyzés: $\pi_\rho(F) \subseteq F^+$ persze mindig igaz.
Ha a felbontás függőségőrző, akkor elég a darabokon ellenőrzni valamit, ami garantálja, hogy F minden függése fennmarad az egészen.

Példa

R(**V**áros, **U**tca, **I**rányítószám) $\quad F = \{VU \to I;\ I \to V\}$

Példa

R(**V**áros, **U**tca, **I**rányítószám) $F = \{VU \to I;\ I \to V\}$

Ez nem BCNF $I \to V$ miatt.

Példa

R(**V**áros, **U**tca, **I**rányítószám) $F = \{VU \to I;\ I \to V\}$

Ez nem BCNF $I \to V$ miatt.

Mire jó a függőségőrzés?:

Ha felbontjuk $\Longrightarrow S(V, I),\ Q(I, U)$

Példa

R(**V**áros, **U**tca, **I**rányítószám) $F = \{VU \to I;\ I \to V\}$

Ez nem BCNF $I \to V$ miatt.

Mire jó a függőségőrzés?:

Ha felbontjuk $\Longrightarrow S(V, I),\ Q(I, U)$

Beszúrunk 2-2 sort:

S	V	I
	Nagykanizsa	8800
	Nagykanizsa	8831

Q	U	I
	Kossuth	8800
	Kossuth	8831

Példa

R(**V**áros, **U**tca, **I**rányítószám) $F = \{VU \to I;\ I \to V\}$

Ez nem BCNF $I \to V$ miatt.

Mire jó a függőségőrzés?:

Ha felbontjuk $\Longrightarrow S(V, I),\ Q(I, U)$

Beszúrunk 2-2 sort:

S	V	I
	Nagykanizsa	8800
	Nagykanizsa	8831

Q	U	I
	Kossuth	8800
	Kossuth	8831

Noha S-ben és Q-ban oké minden, $S \bowtie Q$-ban nem teljesül a $VU \to I$ függés.

Példa

R(**V**áros, **U**tca, **I**rányítószám) $F = \{VU \to I;\ I \to V\}$

Ez nem BCNF $I \to V$ miatt.

Mire jó a függőségőrzés?:

Ha felbontjuk $\Longrightarrow S(V, I),\ Q(I, U)$

Beszúrunk 2-2 sort:

S	V	I
	Nagykanizsa	8800
	Nagykanizsa	8831

Q	U	I
	Kossuth	8800
	Kossuth	8831

Noha S-ben és Q-ban oké minden, $S \bowtie Q$-ban nem teljesül a $VU \to I$ függés.

Ez nem lett volna, ha függőségőrző lenne a felbontás.

Szomorú példa ez: semelyik felbontása sem őrzi meg $VU \to I$-t, mert csak ez olyan függés, aminek jobb oldalán van I, azaz ha egy felbontás függőségőrző lenne, akkor egy tagjában kéne VUI-nek lennie, de az nem lenne valódi felbontás.

Példa

R(**V**áros, **U**tca, **I**rányítószám) $F = \{VU \to I;\ I \to V\}$

Ez nem BCNF $I \to V$ miatt.

Mire jó a függőségőrzés?:

Ha felbontjuk $\Longrightarrow S(V, I),\ Q(I, U)$

Beszúrunk 2-2 sort:

S	V	I
	Nagykanizsa	8800
	Nagykanizsa	8831

Q	U	I
	Kossuth	8800
	Kossuth	8831

Noha S-ben és Q-ban oké minden, $S \bowtie Q$-ban nem teljesül a $VU \to I$ függés.

Ez nem lett volna, ha függőségőrző lenne a felbontás.

Szomorú példa ez: semelyik felbontása sem őrzi meg $VU \to I$-t, mert csak ez olyan függés, aminek jobb oldalán van I, azaz ha egy felbontás függőségőrző lenne, akkor egy tagjában kéne VUI-nek lennie, de az nem lenne valódi felbontás.

$\Longrightarrow$ ennek nincs függőségőrző valódi felbontása, vagyis van olyan reláció, amit nem lehet függőségőrző módon BCNF-ekre szétszedni

Következmény

Állítás

Felbontás BCNF-be nem feltétlenül függőségőrző.

Következmény

Állítás

Felbontás BCNF-be nem feltétlenül függőségőrző.

Kellene egy gyengébb normálforma. Ebben lehet valamennyi redundancia, de legyen függőségőrző.

Adatbázisok elmélete

3NF

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

15. előadás

Definíció

*Az (R, F) séma A attribútuma **prím** (elsődleges), ha szerepel valamelyik kulcsban.*

Definíció

*Az (R, F) séma A attribútuma **prím** (elsődleges), ha szerepel valamelyik kulcsban.*

Szuperkulcsban minden szerepel, kulcs helyett szuperkulccsal nem lenne sok értelme az előbbi definíciónak.

3NF

Definíció

*Az (R, F) séma A attribútuma **prím** (elsődleges), ha szerepel valamelyik kulcsban.*

Szuperkulcsban minden szerepel, kulcs helyett szuperkulccsal nem lenne sok értelme az előbbi definíciónak.

Definíció

*Az (R, F) séma **3NF** (harmadik normálformájú), ha tetszőleges nemtriviális $X \to A \in F^+$ függés esetén vagy X szuperkulcs vagy A prímattribútum.*

3NF

Definíció

*Az (R, F) séma A attribútuma **prím** (elsődleges), ha szerepel valamelyik kulcsban.*

Szuperkulcsban minden szerepel, kulcs helyett szuperkulccsal nem lenne sok értelme az előbbi definíciónak.

Definíció

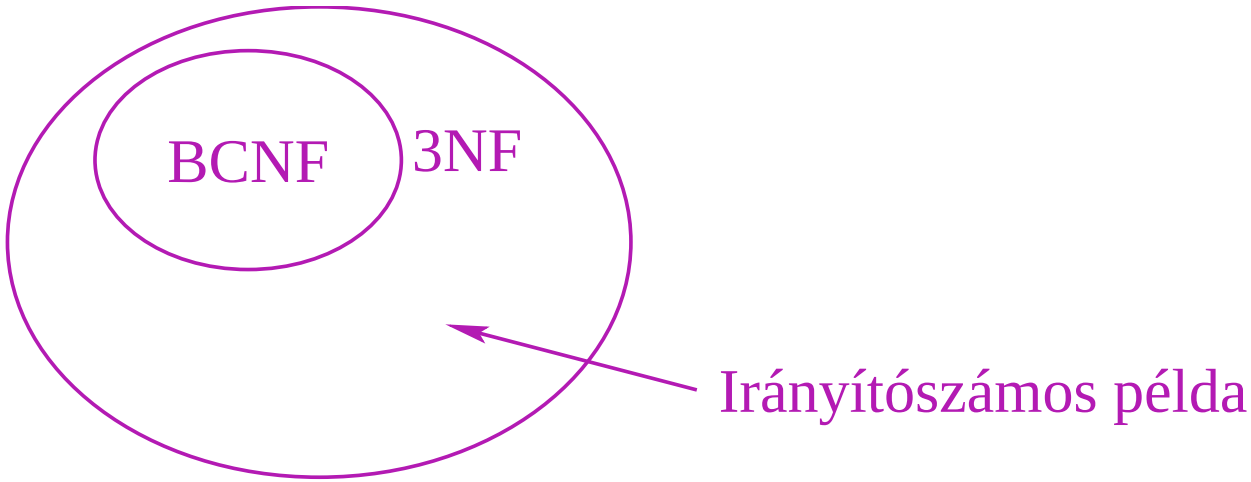
*Az (R, F) séma **3NF** (harmadik normálformájú), ha tetszőleges nemtriviális $X \to A \in F^+$ függés esetén vagy X szuperkulcs vagy A prímattribútum.*

Következmény

Minden BCNF séma egyben 3NF is.

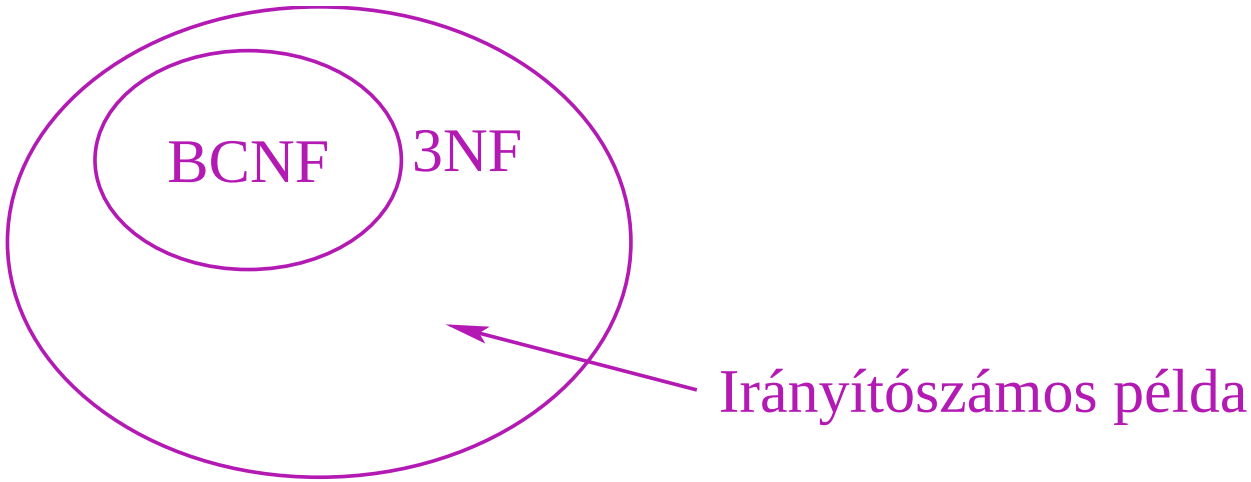
3NF

3NF lehet redundáns, de nem nagyon.



3NF

3NF lehet redundáns, de nem nagyon.

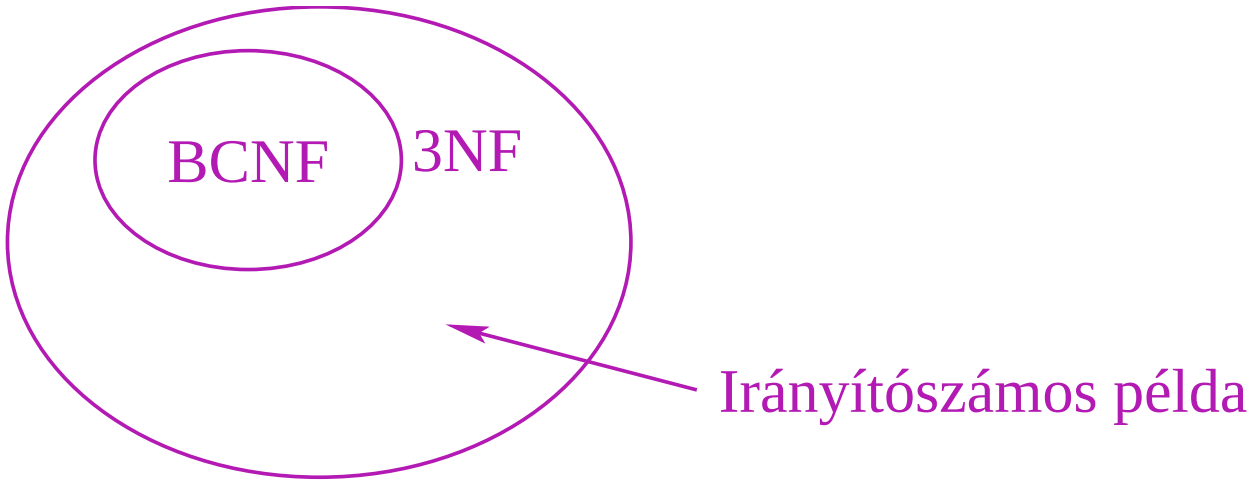


Tétel

Ha (R, F) egy 3NF séma, akkor minden nem prím A attribútumra és $X \subseteq R$ kulcsra igaz, hogy nincs olyan Y, hogy $X \to Y$, $Y \nrightarrow X$, $Y \to A$ és $A \notin Y$. (Nem-elsődleges attribútum nem függ tranzitívan kulcstól.)

3NF

3NF lehet redundáns, de nem nagyon.



Tétel

Ha (R, F) egy 3NF séma, akkor minden nem prím A attribútumra és $X \subseteq R$ kulcsra igaz, hogy nincs olyan Y, hogy $X \to Y$, $Y \nrightarrow X$, $Y \to A$ és $A \notin Y$. (Nem-elsődleges attribútum nem függ tranzitívan kulcstól.)

Nem bizonyítjuk, úgy menne, mint BCNF-nél a hasonló állítás.

3NF tulajdonság ellenőrzése

Hogyan tudjuk ellenőrizni, hogy egy séma 3NF-ben van-e?

3NF tulajdonság ellenőrzése

Hogyan tudjuk ellenőrizni, hogy egy séma 3NF-ben van-e?

Állítás

$\leq$ 2 attribútumos reláció mindig 3NF.

3NF tulajdonság ellenőrzése

Hogyan tudjuk ellenőrizni, hogy egy séma 3NF-ben van-e?

Állítás

$\leq$ 2 attribútumos reláció mindig 3NF.

Bizonyítás.

Már láttuk, hogy BCNF $\Longrightarrow$ 3NF

3NF tulajdonság ellenőrzése

Hogyan tudjuk ellenőrizni, hogy egy séma 3NF-ben van-e?

Állítás

$\leq$ 2 attribútumos reláció mindig 3NF.

Bizonyítás.

Már láttuk, hogy BCNF $\Longrightarrow$ 3NF $\square$

Most is F^+ összes függőségét végig kellene nézni.

3NF tulajdonság ellenőrzése

Hogyan tudjuk ellenőrizni, hogy egy séma 3NF-ben van-e?

Állítás

$\leq$ 2 attribútumos reláció mindig 3NF.

Bizonyítás.

Már láttuk, hogy BCNF $\Longrightarrow$ 3NF $\square$

Most is F^+ összes függőségét végig kellene nézni.
DE:

3NF tulajdonság ellenőrzése

Tétel

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

3NF tulajdonság ellenőrzése

Tétel

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

Nem bizonyítjuk.

3NF tulajdonság ellenőrzése

Tétel

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

Nem bizonyítjuk.

Azt tudjuk ellenőrizni egy adott $X \to A$ függésre, hogy X szuperkulcs-e: kiszámítjuk $X^+(F)$-et.

3NF tulajdonság ellenőrzése

Tétel

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

Nem bizonyítjuk.

Azt tudjuk ellenőrizni egy adott $X \to A$ függésre, hogy X szuperkulcs-e: kiszámítjuk $X^+(F)$-et.
De hogyan ellenőrizzük, hogy A prím-e?

3NF tulajdonság ellenőrzése

Tétel

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

Nem bizonyítjuk.

Azt tudjuk ellenőrizni egy adott $X \to A$ függésre, hogy X szuperkulcs-e: kiszámítjuk $X^+(F)$-et.
De hogyan ellenőrizzük, hogy A prím-e? $\Longrightarrow$ kell az **összes** kulcs

3NF tulajdonság ellenőrzése

Tétel

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

Nem bizonyítjuk.

Azt tudjuk ellenőrizni egy adott $X \to A$ függésre, hogy X szuperkulcs-e: kiszámítjuk $X^+(F)$-et.
De hogyan ellenőrizzük, hogy A prím-e? $\Longrightarrow$ kell az **összes** kulcs

Tétel

Annak eldöntése, hogy egy attribútum prím-e, NP-teljes probléma.

3NF tulajdonság ellenőrzése

Tétel

Ha (R, F) nem 3NF, akkor van olyan $X \to Y \in \mathbf{F}$, amely jobboldalának valamely A attribútumára $X \to A$ nemtriviális, X nem szuperkulcs és A nem prím. (Az ilyen $X \to A \in F^+$.)

Nem bizonyítjuk.

Azt tudjuk ellenőrizni egy adott $X \to A$ függésre, hogy X szuperkulcs-e: kiszámítjuk $X^+(F)$-et.
De hogyan ellenőrizzük, hogy A prím-e? $\Longrightarrow$ kell az **összes** kulcs

Tétel

Annak eldöntése, hogy egy attribútum prím-e, NP-teljes probléma.

Következmény

Annak eldöntése, hogy egy séma 3NF-ben van-e, NP-teljes probléma.

3NF tulajdonság ellenőrzése

Persze olyan algoritmus van, ami legrosszabb esetben exponenciális:

3NF tulajdonság ellenőrzése

Persze olyan algoritmus van, ami legrosszabb esetben exponenciális:

- Meghatározzuk az összes kulcsot.

3NF tulajdonság ellenőrzése

Persze olyan algoritmus van, ami legrosszabb esetben exponenciális:

- Meghatározzuk az összes kulcsot.
- Meghatározzuk az összes prímattribútumot.

3NF tulajdonság ellenőrzése

Persze olyan algoritmus van, ami legrosszabb esetben exponenciális:

- Meghatározzuk az összes kulcsot.
- Meghatározzuk az összes prímattribútumot.
- Minden F-beli $X \to Y$ függésre nézzük meg:
  - Igaz-e, hogy $Y \subseteq X$. Ha igen, a függés triviális. $\surd$

3NF tulajdonság ellenőrzése

Persze olyan algoritmus van, ami legrosszabb esetben exponenciális:

- Meghatározzuk az összes kulcsot.
- Meghatározzuk az összes prímattribútumot.
- Minden F-beli $X \to Y$ függésre nézzük meg:
  - Igaz-e, hogy $Y \subseteq X$. Ha igen, a függés triviális. $\surd$
  - Igaz-e, hogy X kulcs-e. Ha igen, nem sérti a feltételt. $\surd$

3NF tulajdonság ellenőrzése

Persze olyan algoritmus van, ami legrosszabb esetben exponenciális:

- Meghatározzuk az összes kulcsot.
- Meghatározzuk az összes prímattribútumot.
- Minden F-beli $X \to Y$ függésre nézzük meg:
  - Igaz-e, hogy $Y \subseteq X$. Ha igen, a függés triviális. $\surd$
  - Igaz-e, hogy X kulcs-e. Ha igen, nem sérti a feltételt. $\surd$
  - Igaz-e, hogy Y-ben csak prímattribútumok vannak. Ha igen, nem sérti a feltételt. $\surd$

3NF tulajdonság ellenőrzése

Persze olyan algoritmus van, ami legrosszabb esetben exponenciális:

- Meghatározzuk az összes kulcsot.
- Meghatározzuk az összes prímattribútumot.
- Minden F-beli $X \to Y$ függésre nézzük meg:
  - Igaz-e, hogy $Y \subseteq X$. Ha igen, a függés triviális. $\surd$
  - Igaz-e, hogy X kulcs-e. Ha igen, nem sérti a feltételt. $\surd$
  - Igaz-e, hogy Y-ben csak prímattribútumok vannak. Ha igen, nem sérti a feltételt. $\surd$

  Ha egyik sem, akkor van olyan függés, ami sérti a feltételt $\Longrightarrow$ nem 3NF

3NF felbontás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

3NF felbontás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Definíció

*A G függéshalmaz az F függéshalmaz **fedése** ha $G^+ = F^+$. (Persze ilyenkor F is fedése G-nek.)*

3NF felbontás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Definíció

*A G függéshalmaz az F függéshalmaz **fedése** ha $G^+ = F^+$. (Persze ilyenkor F is fedése G-nek.)*

Definíció

*A G függéshalmaz az F függéshalmaz **minimális fedése**, ha egyrészt fedése, másrészt*

(1) *a G-beli függések $X \to A$ alakúak, ahol $A \notin X$*

3NF felbontás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Definíció

*A G függéshalmaz az F függéshalmaz **fedése** ha $G^+ = F^+$. (Persze ilyenkor F is fedése G-nek.)*

Definíció

*A G függéshalmaz az F függéshalmaz **minimális fedése**, ha egyrészt fedése, másrészt*

(1) *a G-beli függések $X \to A$ alakúak, ahol $A \notin X$*

(2) *G-ből nem hagyható el függés: $(G \setminus \{X \to A\})^+ \subsetneq G^+$*

3NF felbontás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Definíció

*A G függéshalmaz az F függéshalmaz **fedése** ha $G^+ = F^+$. (Persze ilyenkor F is fedése G-nek.)*

Definíció

*A G függéshalmaz az F függéshalmaz **minimális fedése**, ha egyrészt fedése, másrészt*

(1) *a G-beli függések $X \to A$ alakúak, ahol $A \notin X$*

(2) *G-ből nem hagyható el függés: $(G \setminus \{X \to A\})^+ \subsetneq G^+$*

(3) *G-beli függések baloldalai minimálisak: $Y \subsetneq X$*
$\Longrightarrow (G \setminus \{X \to A\} \cup \{Y \to A\})^+ \subsetneq G^+$

3NF felbontás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Definíció

*A G függéshalmaz az F függéshalmaz **fedése** ha $G^+ = F^+$. (Persze ilyenkor F is fedése G-nek.)*

Definíció

*A G függéshalmaz az F függéshalmaz **minimális fedése**, ha egyrészt fedése, másrészt*

(1) *a G-beli függések $X \to A$ alakúak, ahol $A \notin X$*

(2) *G-ből nem hagyható el függés: $(G \setminus \{X \to A\})^+ \subsetneq G^+$*

(3) *G-beli függések baloldalai minimálisak: $Y \subsetneq X$*
$\Longrightarrow (G \setminus \{X \to A\} \cup \{Y \to A\})^+ \subsetneq G^+$

3NF felbontás

Állítás

Tetszőleges F-nek van minimális fedése.

3NF felbontás

Állítás

Tetszőleges F-nek van minimális fedése.

Bizonyítás.

Algoritmust adunk, külön gondoskodunk minden pont teljesítéséről.

3NF felbontás

Állítás

Tetszőleges F-nek van minimális fedése.

Bizonyítás.

Algoritmust adunk, külön gondoskodunk minden pont teljesítéséről.
(1) $X \to Y \in G,\ Y = A_1 \dots A_k \implies$ minden $X \to A_i$-t beveszünk, ha $A_i \notin X$.

3NF felbontás

Állítás

Tetszőleges F-nek van minimális fedése.

Bizonyítás.

Algoritmust adunk, külön gondoskodunk minden pont teljesítéséről.
(1) $X \to Y \in G,\ Y = A_1 \dots A_k \implies$ minden $X \to A_i$-t beveszünk, ha $A_i \notin X$.
(2) Minden $X \to A \in G$ függésre kiszámoljuk $Y := X^+(G \setminus \{X \to A\})$-t. Ha $A \in Y$, akkor $X \to A$ elhagyható, különben nem.

3NF felbontás

Állítás

Tetszőleges F-nek van minimális fedése.

Bizonyítás.

Algoritmust adunk, külön gondoskodunk minden pont teljesítéséről.
(1) $X \to Y \in G,\ Y = A_1 \dots A_k \implies$ minden $X \to A_i$-t beveszünk, ha $A_i \notin X$.
(2) Minden $X \to A \in G$ függésre kiszámoljuk $Y := X^+(G \setminus \{X \to A\})$-t. Ha $A \in Y$, akkor $X \to A$ elhagyható, különben nem.
(3) Ellenőrizni kell, hogy $X \to A$ baloldala minimális-e. X minden B elemére kiszámoljuk $Y := (X \setminus \{B\})^+(G)$-t. Ha $A \in Y$, akkor $X \to A$ helyett vegyük be $X - \{B\} \to A$-t. Ha egyik B-re se lesz ilyen, akkor X minimális.

3NF felbontás

Állítás

Tetszőleges F-nek van minimális fedése.

Bizonyítás.

Algoritmust adunk, külön gondoskodunk minden pont teljesítéséről.
(1) $X \to Y \in G,\ Y = A_1 \dots A_k \implies$ minden $X \to A_i$-t beveszünk, ha $A_i \notin X$.
(2) Minden $X \to A \in G$ függésre kiszámoljuk $Y := X^+(G \setminus \{X \to A\})$-t. Ha $A \in Y$, akkor $X \to A$ elhagyható, különben nem.
(3) Ellenőrizni kell, hogy $X \to A$ baloldala minimális-e. X minden B elemére kiszámoljuk $Y := (X \setminus \{B\})^+(G)$-t. Ha $A \in Y$, akkor $X \to A$ helyett vegyük be $X - \{B\} \to A$-t. Ha egyik B-re se lesz ilyen, akkor X minimális.
Megjegyzés: És persze a fenti három lépés során a függéshalmaz lezártja nem változik. □

Példa

$R = (A, B, C, D)$ $\quad F = \{AB \to CD;\ AC \to BD;\ C \to A;\ C \to B\}$

Példa

$R = (A, B, C, D)$ $\quad F = \{AB \to CD;\ AC \to BD;\ C \to A;\ C \to B\}$

(1) $F' = \{AB \to C;\ AB \to D;\ AC \to B;\ AC \to D;\ C \to A;\ C \to B\}$

Példa

$R = (A, B, C, D)$ $\quad F = \{AB \to CD;\ AC \to BD;\ C \to A;\ C \to B\}$

(1) $F' = \{AB \to C;\ AB \to D;\ AC \to B;\ AC \to D;\ C \to A;\ C \to B\}$

(2) $C \to B$ miatt $AC \to B$ elhagyható és $AB \to C$ és $AC \to D$ miatt $AB \to D$ elhagyható, de más nem, ezt végig lehet nézni.

$F' = \{AB \to C;\ AC \to D;\ C \to A;\ C \to B\}$

Példa

$R = (A, B, C, D)$ $\quad F = \{AB \to CD;\ AC \to BD;\ C \to A;\ C \to B\}$

(1) $F' = \{AB \to C;\ AB \to D;\ AC \to B;\ AC \to D;\ C \to A;\ C \to B\}$

(2) $C \to B$ miatt $AC \to B$ elhagyható és $AB \to C$ és $AC \to D$ miatt $AB \to D$ elhagyható, de más nem, ezt végig lehet nézni.

$F' = \{AB \to C;\ AC \to D;\ C \to A;\ C \to B\}$

(3) $C \to A$ miatt $AC \to D$ baloldaláról A elhagyható.

$F'' = \{AB \to C;\ C \to D;\ C \to A;\ C \to B\}$

Példa

$R = (A, B, C, D)$ $\quad F = \{AB \to CD;\ AC \to BD;\ C \to A;\ C \to B\}$

(1) $F' = \{AB \to C;\ AB \to D;\ AC \to B;\ AC \to D;\ C \to A;\ C \to B\}$

(2) $C \to B$ miatt $AC \to B$ elhagyható és $AB \to C$ és $AC \to D$ miatt $AB \to D$ elhagyható, de más nem, ezt végig lehet nézni.

$F' = \{AB \to C;\ AC \to D;\ C \to A;\ C \to B\}$

(3) $C \to A$ miatt $AC \to D$ baloldaláról A elhagyható.

$F'' = \{AB \to C;\ C \to D;\ C \to A;\ C \to B\}$

Ez már minimális fedés.

Példa

$R = (A, B, C, D)$ $\quad F = \{AB \to CD;\ AC \to BD;\ C \to A;\ C \to B\}$

(1) $F' = \{AB \to C;\ AB \to D;\ AC \to B;\ AC \to D;\ C \to A;\ C \to B\}$

(2) $C \to B$ miatt $AC \to B$ elhagyható és $AB \to C$ és $AC \to D$ miatt $AB \to D$ elhagyható, de más nem, ezt végig lehet nézni.

$F' = \{AB \to C;\ AC \to D;\ C \to A;\ C \to B\}$

(3) $C \to A$ miatt $AC \to D$ baloldaláról A elhagyható.

$F'' = \{AB \to C;\ C \to D;\ C \to A;\ C \to B\}$

Ez már minimális fedés.

A minimális fedés nem feltétlenül egyértelmű!

Példa

$R = (A, B, C, D)$ $\quad F = \{AB \to CD;\ AC \to BD;\ C \to A;\ C \to B\}$

(1) $F' = \{AB \to C;\ AB \to D;\ AC \to B;\ AC \to D;\ C \to A;\ C \to B\}$

(2) $C \to B$ miatt $AC \to B$ elhagyható és $AB \to C$ és $AC \to D$ miatt $AB \to D$ elhagyható, de más nem, ezt végig lehet nézni.

$F' = \{AB \to C;\ AC \to D;\ C \to A;\ C \to B\}$

(3) $C \to A$ miatt $AC \to D$ baloldaláról A elhagyható.

$F'' = \{AB \to C;\ C \to D;\ C \to A;\ C \to B\}$

Ez már minimális fedés.

A minimális fedés nem feltétlenül egyértelmű!

Példa: $R(A, B, C)$ $\quad F = \{AB \to C;\ A \to B;\ B \to A\}$ esetén jó minimális fedés lesz

$G_1 = \{B \to C;\ A \to B;\ B \to A\}$ és

$G_2 = \{A \to C;\ A \to B;\ B \to A\}$ is.

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF
Többi R_i is 3NF: tegyük fel, hogy nem az $\Longrightarrow \exists\ U \to B$ nemtriviális függés, hogy U nem szuperkulcs R_i-ben és B nem prímattribútum R_i-ben.
Ha $B = A_i$, akkor $U \subseteq X_i$, de $U \neq X_i$, hiszen akkor U szuperkulcs lenne R_i-ben.

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF
Többi R_i is 3NF: tegyük fel, hogy nem az $\Longrightarrow \exists\ U \to B$ nemtriviális függés, hogy U nem szuperkulcs R_i-ben és B nem prímattribútum R_i-ben.
Ha $B = A_i$, akkor $U \subseteq X_i$, de $U \neq X_i$, hiszen akkor U szuperkulcs lenne R_i-ben.
$\Longrightarrow U \subset X_i \Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben U-ra.

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF
Többi R_i is 3NF: tegyük fel, hogy nem az $\Longrightarrow \exists\ U \to B$ nemtriviális függés, hogy U nem szuperkulcs R_i-ben és B nem prímattribútum R_i-ben.
Ha $B = A_i$, akkor $U \subseteq X_i$, de $U \neq X_i$, hiszen akkor U szuperkulcs lenne R_i-ben.
$\Longrightarrow U \subset X_i \Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben U-ra. Ellentmondás, mert akkor G nem volt minimális fedés.

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF
Többi R_i is 3NF: tegyük fel, hogy nem az $\Longrightarrow \exists\ U \to B$ nemtriviális függés, hogy U nem szuperkulcs R_i-ben és B nem prímattribútum R_i-ben.
Ha $B = A_i$, akkor $U \subseteq X_i$, de $U \neq X_i$, hiszen akkor U szuperkulcs lenne R_i-ben.
$\Longrightarrow U \subset X_i \Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben U-ra. Ellentmondás, mert akkor G nem volt minimális fedés.
Ha $B \neq A_i \Longrightarrow B \in X_i$ és B nem prím R_i-ben $\Longrightarrow X_i$ nem kulcs R_i-ben (de szuperkulcs)

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF
Többi R_i is 3NF: tegyük fel, hogy nem az $\Longrightarrow \exists\ U \to B$ nemtriviális függés, hogy U nem szuperkulcs R_i-ben és B nem prímattribútum R_i-ben.
Ha $B = A_i$, akkor $U \subseteq X_i$, de $U \neq X_i$, hiszen akkor U szuperkulcs lenne R_i-ben.
$\Longrightarrow U \subset X_i \Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben U-ra. Ellentmondás, mert akkor G nem volt minimális fedés.
Ha $B \neq A_i \Longrightarrow B \in X_i$ és B nem prím R_i-ben $\Longrightarrow X_i$ nem kulcs R_i-ben (de szuperkulcs) $\Longrightarrow \exists\ Y \subsetneq X_i$ kulcs R_i-ben $\Longrightarrow Y \to A_i$ fennáll

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF
Többi R_i is 3NF: tegyük fel, hogy nem az $\Longrightarrow \exists\ U \to B$ nemtriviális függés, hogy U nem szuperkulcs R_i-ben és B nem prímattribútum R_i-ben.
Ha $B = A_i$, akkor $U \subseteq X_i$, de $U \neq X_i$, hiszen akkor U szuperkulcs lenne R_i-ben.
$\Longrightarrow U \subset X_i \Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben U-ra. Ellentmondás, mert akkor G nem volt minimális fedés.
Ha $B \neq A_i \Longrightarrow B \in X_i$ és B nem prím R_i-ben $\Longrightarrow X_i$ nem kulcs R_i-ben (de szuperkulcs) $\Longrightarrow \exists\ Y \subsetneq X_i$ kulcs R_i-ben $\Longrightarrow Y \to A_i$ fennáll $\Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben Y-ra, megint csak ellentmondás.

Bizonyítás

Tétel

Tetszőleges (R, F) sémának van hűséges és függőségőrző felbontása 3NF sémákra.

Bizonyítás.

Vegyük F egy minimális fedését: $G = \{X_1 \to A_1, \ldots, X_k \to A_k\}$
Legyen X egy kulcs és $\rho = (X, X_1A_1, \ldots, X_kA_k)$ egy felbontás. $\Longrightarrow R_0, R_1, \ldots, R_k$
Állítás: ez függőségőrző lesz, a tagok 3NF-ek és a felbontás hűséges.
ρ **függőségőrző:** $F^+ = G^+$ és minden G-beli $X \to A_i$ függés benne lesz R_i-ben (ott ellenőrizhető).
R_0 **3NF:** R_0-ban nincs nemtriviális függés, mert különben X nem lenne kulcs, csak szuperkulcs $\Longrightarrow R_0$ BCNF $\Longrightarrow$ 3NF
Többi R_i is 3NF: tegyük fel, hogy nem az $\Longrightarrow \exists\ U \to B$ nemtriviális függés, hogy U nem szuperkulcs R_i-ben és B nem prímattribútum R_i-ben.
Ha $B = A_i$, akkor $U \subseteq X_i$, de $U \neq X_i$, hiszen akkor U szuperkulcs lenne R_i-ben. $\Longrightarrow U \subset X_i \Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben U-ra. Ellentmondás, mert akkor G nem volt minimális fedés.
Ha $B \neq A_i \Longrightarrow B \in X_i$ és B nem prím R_i-ben $\Longrightarrow X_i$ nem kulcs R_i-ben (de szuperkulcs) $\Longrightarrow \exists\ Y \subsetneq X_i$ kulcs R_i-ben $\Longrightarrow Y \to A_i$ fennáll $\Longrightarrow X_i \to A_i$ baloldala csökkenthető G-ben Y-ra, megint csak ellentmondás.
ρ **hűséges:** Higgyük el, nem bizonyítjuk. □

Megjegyzés: Előfordulhat, hogy valamelyik X_iA_i már tartalmaz kulcsot. Ilyenkor a $\rho = \{X_1A_1, \ldots, X_kA_k\}$ is jó felbontás már.

Megjegyzés: Előfordulhat, hogy valamelyik $X_i A_i$ már tartalmaz kulcsot. Ilyenkor a $\rho = \{X_1 A_1, \ldots, X_k A_k\}$ is jó felbontás már.

Megjegyzés: 2NF már nem érdekes, 1NF kicsit érdekes, de nem foglalkozunk vele.

Példa: 3NF-re bontás

$R = (A, B, C, D, E)$ $\quad F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$

Példa: 3NF-re bontás

$R = (A, B, C, D, E)$ $F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$

Ez nem 3NF, mert a kulcsok:

semelyik egyelemű halmaz nem kulcs (csak D lehetne, de az ő lezártja csak DE),

Példa: 3NF-re bontás

$R = (A, B, C, D, E)$ $F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$
Ez nem 3NF, mert a kulcsok:
semelyik egyelemű halmaz nem kulcs (csak D lehetne, de az ő lezártja csak DE),
viszont kételeműek közül szuperkulcs lesz AC, AD, AE (A-nak benne kell lennie minden kulcsban, mert A nincs jobboldalon), AB viszont nem szuperkulcs.

Példa: 3NF-re bontás

$R = (A, B, C, D, E)$ $F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$
Ez nem 3NF, mert a kulcsok:
semelyik egyelemű halmaz nem kulcs (csak D lehetne, de az ő lezártja csak DE),
viszont kételeműek közül szuperkulcs lesz AC, AD, AE (A-nak benne kell lennie minden kulcsban, mert A nincs jobboldalon), AB viszont nem szuperkulcs.
Ezek kulcsok is lesznek, mert egyik egyelemű se volt kulcs.

Példa: 3NF-re bontás

$R = (A, B, C, D, E)$ $F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$
Ez nem 3NF, mert a kulcsok:
semelyik egyelemű halmaz nem kulcs (csak D lehetne, de az ő lezártja csak DE),
viszont kételeműek közül szuperkulcs lesz AC, AD, AE (A-nak benne kell lennie minden kulcsban, mert A nincs jobboldalon), AB viszont nem szuperkulcs.
Ezek kulcsok is lesznek, mert egyik egyelemű se volt kulcs.
Más kulcs nincs is, mert ha lenne legalább háromelemű halmaz, aminek a lezártja az egész, akkor abban A biztos benne van és legalább C vagy D vagy E is benne van, de akkor az már csak szuperkulcs lehet, mert tartalmaz kulcsot.

Példa: 3NF-re bontás

$R = (A, B, C, D, E)$ $F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$
Ez nem 3NF, mert a kulcsok:
semelyik egyelemű halmaz nem kulcs (csak D lehetne, de az ő lezártja csak DE),
viszont kételeműek közül szuperkulcs lesz AC, AD, AE (A-nak benne kell lennie minden kulcsban, mert A nincs jobboldalon), AB viszont nem szuperkulcs.
Ezek kulcsok is lesznek, mert egyik egyelemű se volt kulcs.
Más kulcs nincs is, mert ha lenne legalább háromelemű halmaz, aminek a lezártja az egész, akkor abban A biztos benne van és legalább C vagy D vagy E is benne van, de akkor az már csak szuperkulcs lehet, mert tartalmaz kulcsot.

Innen látszik, hogy a prímattribútumok: A, C, D, E, vagyis B nem az.

Példa: 3NF-re bontás (folyt.)

$R = (A, B, C, D, E)$ $\quad F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$

Tehát a $CD \to B$ függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím.

Példa: 3NF-re bontás (folyt.)

$R = (A, B, C, D, E)$ $\quad F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$

Tehát a $CD \to B$ függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím.

Csináljunk hát egy 3NF-ekre való függőségőrző, hűséges felbontást.

Példa: 3NF-re bontás (folyt.)

$R = (A, B, C, D, E)$ $\quad F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$

Tehát a $CD \to B$ függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím.

Csináljunk hát egy 3NF-ekre való függőségőrző, hűséges felbontást.

(1) $F' = \{AE \to B;\ AE \to C;\ AC \to D;\ CD \to B;\ CD \to E;\ D \to E\}$

Példa: 3NF-re bontás (folyt.)

$R = (A, B, C, D, E)$ $\quad F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$

Tehát a $CD \to B$ függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím.

Csináljunk hát egy 3NF-ekre való függőségőrző, hűséges felbontást.

(1) $F' = \{AE \to B;\ AE \to C;\ AC \to D;\ CD \to B;\ CD \to E;\ D \to E\}$

(2) $AE \to C, AC \to D, CD \to B$ miatt $AE \to B$ elhagyható és $D \to E$ miatt $CD \to E$ is elhagyható,

Példa: 3NF-re bontás (folyt.)

$R = (A, B, C, D, E)$ $\quad F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$

Tehát a $CD \to B$ függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím.

Csináljunk hát egy 3NF-ekre való függőségőrző, hűséges felbontást.

(1) $F' = \{AE \to B;\ AE \to C;\ AC \to D;\ CD \to B;\ CD \to E;\ D \to E\}$

(2) $AE \to C, AC \to D, CD \to B$ miatt $AE \to B$ elhagyható és $D \to E$ miatt $CD \to E$ is elhagyható, de más nem, ezt végig lehet nézni (mert például AE-nek a maradék függésekre vett lezártjában nincsen benne C).

$F'' = \{AE \to C;\ AC \to D;\ CD \to B;\ D \to E\}$

Példa: 3NF-re bontás (folyt.)

$R = (A, B, C, D, E)$ $F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$
Tehát a $CD \to B$ függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím.
Csináljunk hát egy 3NF-ekre való függőségőrző, hűséges felbontást.
(1) $F' = \{AE \to B;\ AE \to C;\ AC \to D;\ CD \to B;\ CD \to E;\ D \to E\}$
(2) $AE \to C, AC \to D, CD \to B$ miatt $AE \to B$ elhagyható és $D \to E$ miatt $CD \to E$ is elhagyható, de más nem, ezt végig lehet nézni (mert például AE-nek a maradék függésekre vett lezártjában nincsen benne C).
$F'' = \{AE \to C;\ AC \to D;\ CD \to B;\ D \to E\}$
(3) Semelyik baloldal nem csökkenthető, mert például A lezártjában nincsen benne E, és a többi is ugyanígy látszik. Vagyis F'' már minimális fedés.

Példa: 3NF-re bontás (folyt.)

$R = (A, B, C, D, E)$ $F = \{AE \to BC;\ AC \to D;\ CD \to BE;\ D \to E\}$
Tehát a $CD \to B$ függés rossz a 3NF szempontjából, mert CD nem szuperkulcs és B nem prím.
Csináljunk hát egy 3NF-ekre való függőségőrző, hűséges felbontást.
(1) $F' = \{AE \to B;\ AE \to C;\ AC \to D;\ CD \to B;\ CD \to E;\ D \to E\}$
(2) $AE \to C, AC \to D, CD \to B$ miatt $AE \to B$ elhagyható és $D \to E$ miatt $CD \to E$ is elhagyható, de más nem, ezt végig lehet nézni (mert például AE-nek a maradék függésekre vett lezártjában nincsen benne C).
$F'' = \{AE \to C;\ AC \to D;\ CD \to B;\ D \to E\}$
(3) Semelyik baloldal nem csökkenthető, mert például A lezártjában nincsen benne E, és a többi is ugyanígy látszik. Vagyis F'' már minimális fedés.
A minimális fedés alapján a jó felbontás:
(AEC, ACD, CDB, DE) mivel kulcsot nem is kellett hozzávennünk, mert az már benne van az egyik tagban (pl. AE az elsőben).

Többértékű függés

A legfontosabb a funkcionális függés, de vannak másféle függések is.

Többértékű függés

A legfontosabb a funkcionális függés, de vannak másféle függések is.
Motiváló példa: R(Név, Tantárgy, Gyereknév)

Név	Tantárgy	Gyereknév
Katona	Algel	Dani
Katona	Adatbázis	Lilla
Katona	Algel	Lilla
Katona	Adatbázis	Dani

Többértékű függés

A legfontosabb a funkcionális függés, de vannak másféle függések is.
Motiváló példa: R(Név, Tantárgy, Gyereknév)

Név	Tantárgy	Gyereknév
Katona	Algel	Dani
Katona	Adatbázis	Lilla
Katona	Algel	Lilla
Katona	Adatbázis	Dani

Ez BCNF, de mégis redundáns, mert ha valamelyik tárgynál szerepel egy gyereknév, akkor az összes többinél is szerepelnie kell. (Pl. beszúrni nehéz, mert amikor egy sort beszúrok, figyelni kell arra, hogy egy másikat is beszúrjak.)

Többértékű függés

A legfontosabb a funkcionális függés, de vannak másféle függések is.
Motiváló példa: R(Név, Tantárgy, Gyereknév)

Név	Tantárgy	Gyereknév
Katona	Algel	Dani
Katona	Adatbázis	Lilla
Katona	Algel	Lilla
Katona	Adatbázis	Dani

Ez BCNF, de mégis redundáns, mert ha valamelyik tárgynál szerepel egy gyereknév, akkor az összes többinél is szerepelnie kell. (Pl. beszúrni nehéz, mert amikor egy sort beszúrok, figyelni kell arra, hogy egy másikat is beszúrjak.)
Jobb lenne tárolni (Név, Tantárgy) és (Név, Gyereknév) felbontásban.

Többértékű függés

A legfontosabb a funkcionális függés, de vannak másféle függések is.
Motiváló példa: R(Név, Tantárgy, Gyereknév)

Név	Tantárgy	Gyereknév
Katona	Algel	Dani
Katona	Adatbázis	Lilla
Katona	Algel	Lilla
Katona	Adatbázis	Dani

Ez BCNF, de mégis redundáns, mert ha valamelyik tárgynál szerepel egy gyereknév, akkor az összes többinél is szerepelnie kell. (Pl. beszúrni nehéz, mert amikor egy sort beszúrok, figyelni kell arra, hogy egy másikat is beszúrjak.)
Jobb lenne tárolni (Név, Tantárgy) és (Név, Gyereknév) felbontásban.
Ok: a Tantárgy és a Gyereknév független (minden kombinációban előfordulnak)

Többértékű függés

A legfontosabb a funkcionális függés, de vannak másféle függések is.
Motiváló példa: R(Név, Tantárgy, Gyereknév)

Név	Tantárgy	Gyereknév
Katona	Algel	Dani
Katona	Adatbázis	Lilla
Katona	Algel	Lilla
Katona	Adatbázis	Dani

Ez BCNF, de mégis redundáns, mert ha valamelyik tárgynál szerepel egy gyereknév, akkor az összes többinél is szerepelnie kell. (Pl. beszúrni nehéz, mert amikor egy sort beszúrok, figyelni kell arra, hogy egy másikat is beszúrjak.)
Jobb lenne tárolni (Név, Tantárgy) és (Név, Gyereknév) felbontásban.
Ok: a Tantárgy és a Gyereknév független (minden kombinációban előfordulnak)
$\Longrightarrow$ ha látjuk az első két sort, tudjuk, hogy a másik kettő is ott van.

Adatbázisok elmélete

Többértékű függés, 4NF

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

16. előadás

Többértékű függés

Definíció

*Az X attribútumhalmaztól **többértékűen függ** az Y attribútumhalmaz az r relációban (jele: $X \twoheadrightarrow Y$), ha tetszőleges $t_1, t_2 \in r$ sorokra, melyekre $t_1[X] = t_2[X]$, létezik $t_3, t_4 \in r$, melyekre*

- $t_3[XY] = t_1[XY]$
- $t_3[R \setminus XY] = t_2[R \setminus XY]$
- $t_4[XY] = t_2[XY]$
- $t_4[R \setminus XY] = t_1[R \setminus XY]$

Többértékű függés

Definíció

*Az X attribútumhalmaztól **többértékűen függ** az Y attribútumhalmaz az r relációban (jele: $X \twoheadrightarrow Y$), ha tetszőleges $t_1, t_2 \in r$ sorokra, melyekre $t_1[X] = t_2[X]$, létezik $t_3, t_4 \in r$, melyekre*

- $t_3[XY] = t_1[XY]$
- $t_3[R \setminus XY] = t_2[R \setminus XY]$
- $t_4[XY] = t_2[XY]$
- $t_4[R \setminus XY] = t_1[R \setminus XY]$

	$\overbrace{\quad\quad}^{X}$	$\overbrace{\quad\quad}^{Y}$	$\overbrace{\quad\quad}^{R \setminus XY}$
t_1	*AAAAAAA*	*BBBBBBB*	*CCCCCCC*
t_2	*AAAAAAA*	*DDDDDDD*	*EEEEEEE*

Többértékű függés

Definíció

*Az X attribútumhalmaztól **többértékűen függ** az Y attribútumhalmaz az r relációban (jele: $X \twoheadrightarrow Y$), ha tetszőleges $t_1, t_2 \in r$ sorokra, melyekre $t_1[X] = t_2[X]$, létezik $t_3, t_4 \in r$, melyekre*

- $t_3[XY] = t_1[XY]$
- $t_3[R \setminus XY] = t_2[R \setminus XY]$
- $t_4[XY] = t_2[XY]$
- $t_4[R \setminus XY] = t_1[R \setminus XY]$

	$\overbrace{\quad\quad}^{X}$	$\overbrace{\quad\quad}^{Y}$	$\overbrace{\quad\quad}^{R \setminus XY}$
t_1	*AAAAAAA*	*BBBBBBB*	*CCCCCCC*
t_2	*AAAAAAA*	*DDDDDDD*	*EEEEEEE*
	$\vdots$	$\vdots$	$\vdots$
t_3	*AAAAAAA*	*BBBBBBB*	*EEEEEEE*
t_4	*AAAAAAA*	*DDDDDDD*	*CCCCCCC*

Többértékű függés

Definíció

*Az X attribútumhalmaztól **többértékűen függ** az Y attribútumhalmaz az r relációban (jele: $X \twoheadrightarrow Y$), ha tetszőleges $t_1, t_2 \in r$ sorokra, melyekre $t_1[X] = t_2[X]$, létezik $t_3, t_4 \in r$, melyekre*

- $t_3[XY] = t_1[XY]$
- $t_3[R \setminus XY] = t_2[R \setminus XY]$
- $t_4[XY] = t_2[XY]$
- $t_4[R \setminus XY] = t_1[R \setminus XY]$

	$\overbrace{\quad\quad}^{X}$	$\overbrace{\quad\quad}^{Y}$	$\overbrace{\quad\quad}^{R \setminus XY}$
t_1	*AAAAAAA*	*BBBBBBB*	*CCCCCCC*
t_2	*AAAAAAA*	*DDDDDDD*	*EEEEEEE*
	$\vdots$	$\vdots$	$\vdots$
t_3	*AAAAAAA*	*BBBBBBB*	*EEEEEEE*
t_4	*AAAAAAA*	*DDDDDDD*	*CCCCCCC*

Többértékű függés

Definíció

*Az X attribútumhalmaztól **többértékűen függ** az Y attribútumhalmaz az r relációban (jele: $X \twoheadrightarrow Y$), ha tetszőleges $t_1, t_2 \in r$ sorokra, melyekre $t_1[X] = t_2[X]$, létezik $t_3, t_4 \in r$, melyekre*

- $t_3[XY] = t_1[XY]$
- $t_3[R \setminus XY] = t_2[R \setminus XY]$
- $t_4[XY] = t_2[XY]$
- $t_4[R \setminus XY] = t_1[R \setminus XY]$

	$\overbrace{\quad\quad}^{X}$	$\overbrace{\quad\quad}^{Y}$	$\overbrace{\quad\quad}^{R \setminus XY}$
t_1	*AAAAAAA*	*BBBBBBB*	*CCCCCCC*
t_2	*AAAAAAA*	*DDDDDDD*	*EEEEEEE*
	$\vdots$	$\vdots$	$\vdots$
t_3	*AAAAAAA*	*BBBBBBB*	*EEEEEEE*
t_4	*AAAAAAA*	*DDDDDDD*	*CCCCCCC*

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$,

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_2$ *és* $t_4 = t_1$ *jó lesz.*

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_2$ *és* $t_4 = t_1$ *jó lesz.*
- $XY = R \implies X \twoheadrightarrow Y$,

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_2$ *és* $t_4 = t_1$ *jó lesz.*
- $XY = R \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_1$ *és* $t_4 = t_2$ *jó lesz.*

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_2$ *és* $t_4 = t_1$ *jó lesz.*
- $XY = R \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_1$ *és* $t_4 = t_2$ *jó lesz.*

Ezentúl a többértékű függések is a séma részei lesznek és definiálhatjuk a levezethetőséget ($\vdash$) és a logikai következményt ($\models$) úgy, hogy funkcionális függőségek és többértékű függőségek is vannak F-ben.

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_2$ *és* $t_4 = t_1$ *jó lesz.*
- $XY = R \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_1$ *és* $t_4 = t_2$ *jó lesz.*

Ezentúl a többértékű függések is a séma részei lesznek és definiálhatjuk a levezethetőséget ($\vdash$) és a logikai következményt ($\models$) úgy, hogy funkcionális függőségek és többértékű függőségek is vannak F-ben.
Logikai következmény: egy F (funkcionális és többértékű függéseket is tartalmazó) függéshalmaznak logikai következménye egy (funkcionális vagy többértékű) függés, ha minden olyan relációban, amiben F minden függése fennáll, fenn kell hogy álljon a mondott függés is.

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_2$ *és* $t_4 = t_1$ *jó lesz.*
- $XY = R \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_1$ *és* $t_4 = t_2$ *jó lesz.*

Ezentúl a többértékű függések is a séma részei lesznek és definiálhatjuk a levezethetőséget ($\vdash$) és a logikai következményt ($\models$) úgy, hogy funkcionális függőségek és többértékű függőségek is vannak F-ben.
Logikai következmény: egy F (funkcionális és többértékű függéseket is tartalmazó) függéshalmaznak logikai következménye egy (funkcionális vagy többértékű) függés, ha minden olyan relációban, amiben F minden függése fennáll, fenn kell hogy álljon a mondott függés is.
Levezetés: Armstrong-axiómák (a funkcionális függésekre) és 5 új axióma, amiben $\rightarrow$ és $\twoheadrightarrow$ is van. Amilyen függés ezekkel előáll F-ből, arra mondjuk, hogy levezethető.

Többértékű függések levezetése

Definíció

Triviális többértékű függések *(amik mindig igazak)*:

- $Y \subseteq X \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_2$ *és* $t_4 = t_1$ *jó lesz.*
- $XY = R \implies X \twoheadrightarrow Y$, *mert* $t_3 = t_1$ *és* $t_4 = t_2$ *jó lesz.*

Ezentúl a többértékű függések is a séma részei lesznek és definiálhatjuk a levezethetőséget ($\vdash$) és a logikai következményt ($\models$) úgy, hogy funkcionális függőségek és többértékű függőségek is vannak F-ben.
Logikai következmény: egy F (funkcionális és többértékű függéseket is tartalmazó) függéshalmaznak logikai következménye egy (funkcionális vagy többértékű) függés, ha minden olyan relációban, amiben F minden függése fennáll, fenn kell hogy álljon a mondott függés is.
Levezetés: Armstrong-axiómák (a funkcionális függésekre) és 5 új axióma, amiben $\rightarrow$ és $\twoheadrightarrow$ is van. Amilyen függés ezekkel előáll F-ből, arra mondjuk, hogy levezethető.
Hasonló elmélet, mint $\rightarrow$-nél $\implies$ belátható, hogy $\vdash \sim \models$ itt is igaz lesz.

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$,

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$, mert $t_3 = t_2$ és $t_4 = t_1$ jó lesz.

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$, mert $t_3 = t_2$ és $t_4 = t_1$ jó lesz.
- $X \twoheadrightarrow Y \vdash X \twoheadrightarrow R \setminus XY$,

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$, mert $t_3 = t_2$ és $t_4 = t_1$ jó lesz.
- $X \twoheadrightarrow Y \vdash X \twoheadrightarrow R \setminus XY$, mert $t'_3 = t_4$ és $t'_4 = t_3$ jó lesz.

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$, mert $t_3 = t_2$ és $t_4 = t_1$ jó lesz.
- $X \twoheadrightarrow Y \vdash X \twoheadrightarrow R \setminus XY$, mert $t'_3 = t_4$ és $t'_4 = t_3$ jó lesz.
- De pl. $X \twoheadrightarrow AB \nvdash X \twoheadrightarrow A$, nem szétvágható. (Sok minden máshogy van a többértékű függéseknél.)

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$, mert $t_3 = t_2$ és $t_4 = t_1$ jó lesz.
- $X \twoheadrightarrow Y \vdash X \twoheadrightarrow R \setminus XY$, mert $t_3' = t_4$ és $t_4' = t_3$ jó lesz.
- De pl. $X \twoheadrightarrow AB \nvdash X \twoheadrightarrow A$, nem szétvágható. (Sok minden máshogy van a többértékű függéseknél.)

Tétel

Legyen $\rho = (R_1, R_2)$ az (R, F) séma felbontása, ahol F most funkcionális és többértékű függéseket is tartalmaz. ρ akkor és csak akkor hűséges felbontás, ha $R_1 \cap R_2 \twoheadrightarrow R_2 \setminus R_1$.

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$, mert $t_3 = t_2$ és $t_4 = t_1$ jó lesz.
- $X \twoheadrightarrow Y \vdash X \twoheadrightarrow R \setminus XY$, mert $t_3' = t_4$ és $t_4' = t_3$ jó lesz.
- De pl. $X \twoheadrightarrow AB \nvdash X \twoheadrightarrow A$, nem szétvágható. (Sok minden máshogy van a többértékű függéseknél.)

Tétel

Legyen $\rho = (R_1, R_2)$ az (R, F) séma felbontása, ahol F most funkcionális és többértékű függéseket is tartalmaz. ρ akkor és csak akkor hűséges felbontás, ha $R_1 \cap R_2 \twoheadrightarrow R_2 \setminus R_1$.

Megjegyzés: Nem kell a „vagy $R_1 \cap R_2 \twoheadrightarrow R_1 \setminus R_2$” a fenti 2. szabály miatt, mert ha $R_1 \cap R_2 \twoheadrightarrow R_1 \setminus R_2$ igaz, akkor $R_1 \cap R_2 \twoheadrightarrow R \setminus (R_1 \setminus R_2)$ is igaz, ebből meg már következik $R_1 \cap R_2 \twoheadrightarrow R_2 \setminus R_1$.

Többértékű levezetési szabályok

Két fontos új szabály

- $X \to Y \vdash X \twoheadrightarrow Y$, mert $t_3 = t_2$ és $t_4 = t_1$ jó lesz.
- $X \twoheadrightarrow Y \vdash X \twoheadrightarrow R \setminus XY$, mert $t_3' = t_4$ és $t_4' = t_3$ jó lesz.
- De pl. $X \twoheadrightarrow AB \nvdash X \twoheadrightarrow A$, nem szétvágható. (Sok minden máshogy van a többértékű függéseknél.)

Tétel

Legyen $\rho = (R_1, R_2)$ az (R, F) séma felbontása, ahol F most funkcionális és többértékű függéseket is tartalmaz. ρ akkor és csak akkor hűséges felbontás, ha $R_1 \cap R_2 \twoheadrightarrow R_2 \setminus R_1$.

Megjegyzés: Nem kell a „vagy $R_1 \cap R_2 \twoheadrightarrow R_1 \setminus R_2$” a fenti 2. szabály miatt, mert ha $R_1 \cap R_2 \twoheadrightarrow R_1 \setminus R_2$ igaz, akkor $R_1 \cap R_2 \twoheadrightarrow R \setminus (R_1 \setminus R_2)$ is igaz, ebből meg már következik $R_1 \cap R_2 \twoheadrightarrow R_2 \setminus R_1$.
a tétel bizonyítása hasonló, mint a funkcionális függésnél, de nem bizonyítjuk.

4NF

Cél: olyan normálforma, amiben többértékű függés miatt sincs redundancia.

4NF

Cél: olyan normálforma, amiben többértékű függés miatt sincs redundancia.
BCNF mintájára:

Definíció

*Az (R, F) séma **4NF** (**negyedik normálformájú**), ha tetszőleges nemtriviális $X \twoheadrightarrow Y \in F^+$ esetén X szuperkulcs (a szuperkulcsot a régi értelemben, csak funkcionális függőségekkel definiálva).*

4NF

Cél: olyan normálforma, amiben többértékű függés miatt sincs redundancia.
BCNF mintájára:

Definíció

*Az (R, F) séma **4NF** (**negyedik normálformájú**), ha tetszőleges nemtriviális $X \twoheadrightarrow Y \in F^+$ esetén X szuperkulcs (a szuperkulcsot a régi értelemben, csak funkcionális függőségekkel definiálva).*

Következmény

Ha egy séma 4NF, akkor BCNF is.

Következmény

Ha egy séma 4NF, akkor BCNF is.

Bizonyítás.

Indirekt tegyük fel, hogy létezik olyan $X \to A \in F^+$ nemtriviális függés, ahol X nem szuperkulcs.

Következmény

Ha egy séma 4NF, akkor BCNF is.

Bizonyítás.

Indirekt tegyük fel, hogy létezik olyan $X \to A \in F^+$ nemtriviális függés, ahol X nem szuperkulcs. $\Longrightarrow$ Ekkor  , amiatt, hogy $X \to A$-ból következik, hogy $X \twoheadrightarrow A$.

Következmény

Ha egy séma 4NF, akkor BCNF is.

Bizonyítás.

Indirekt tegyük fel, hogy létezik olyan $X \to A \in F^+$ nemtriviális függés, ahol X nem szuperkulcs. $\Longrightarrow$ Ekkor  , amiatt, hogy $X \to A$-ból következik, hogy $X \twoheadrightarrow A$. □

Megjegyzések:

- Ha F-ben csak funkcionális függőségek vannak, akkor 4NF=BCNF

Következmény

Ha egy séma 4NF, akkor BCNF is.

Bizonyítás.

Indirekt tegyük fel, hogy létezik olyan $X \to A \in F^+$ nemtriviális függés, ahol X nem szuperkulcs. $\Longrightarrow$ Ekkor  , amiatt, hogy $X \to A$-ból következik, hogy $X \twoheadrightarrow A$. □

Megjegyzések:

- Ha F-ben csak funkcionális függőségek vannak, akkor 4NF=BCNF
- 2 attribútumos reláció mindig 4NF, hiszen nincs nemtriviális többértékű függés, azt meg már láttuk, hogy ha csak funkcionális függések vannak, akkor a BCNF-ség rendben van kétattribútumos relációnál.

Következmény

Ha egy séma 4NF, akkor BCNF is.

Bizonyítás.

Indirekt tegyük fel, hogy létezik olyan $X \to A \in F^+$ nemtriviális függés, ahol X nem szuperkulcs. $\Longrightarrow$ Ekkor  , amiatt, hogy $X \to A$-ból következik, hogy $X \twoheadrightarrow A$. □

Megjegyzések:

- Ha F-ben csak funkcionális függőségek vannak, akkor 4NF=BCNF
- 2 attribútumos reláció mindig 4NF, hiszen nincs nemtriviális többértékű függés, azt meg már láttuk, hogy ha csak funkcionális függések vannak, akkor a BCNF-ség rendben van kétattribútumos relációnál.
- Van olyan reláció, ami BCNF, de nem 4NF (a korábbi gyerekes példa, mert ott a Név nem szuperkulcs)

Felbontás 4NF-re

Tétel

Legyen (R, F) egy séma, ahol F funkcionális és többértékű függések halmaza. Ekkor (R, F) felbontható hűségesen 4NF relációkra.

Felbontás 4NF-re

Tétel

Legyen (R, F) egy séma, ahol F funkcionális és többértékű függések halmaza. Ekkor (R, F) felbontható hűségesen 4NF relációkra.

Algoritmus: Hasonlóan BCNF-hez, mindig két valódi részre bontjuk hűségesen, addig, amíg mindegyik rész 4NF nem lesz.

Felbontás 4NF-re

Tétel

Legyen (R, F) egy séma, ahol F funkcionális és többértékű függések halmaza. Ekkor (R, F) felbontható hűségesen 4NF relációkra.

Algoritmus: Hasonlóan BCNF-hez, mindig két valódi részre bontjuk hűségesen, addig, amíg mindegyik rész 4NF nem lesz.
Keresünk egy $X \twoheadrightarrow Y$ függést, ami megsérti a 4NF feltételt.

Felbontás 4NF-re

Tétel

Legyen (R, F) egy séma, ahol F funkcionális és többértékű függések halmaza. Ekkor (R, F) felbontható hűségesen 4NF relációkra.

Algoritmus: Hasonlóan BCNF-hez, mindig két valódi részre bontjuk hűségesen, addig, amíg mindegyik rész 4NF nem lesz.
Keresünk egy $X \twoheadrightarrow Y$ függést, ami megsérti a 4NF feltételt.
Ha van $X \rightarrow Y$, ami megsérti a 4NF feltételt, akkor $X \twoheadrightarrow Y$ is megsérti.

Felbontás 4NF-re

Tétel

Legyen (R, F) egy séma, ahol F funkcionális és többértékű függések halmaza. Ekkor (R, F) felbontható hűségesen 4NF relációkra.

Algoritmus: Hasonlóan BCNF-hez, mindig két valódi részre bontjuk hűségesen, addig, amíg mindegyik rész 4NF nem lesz.
Keresünk egy $X \twoheadrightarrow Y$ függést, ami megsérti a 4NF feltételt.
Ha van $X \rightarrow Y$, ami megsérti a 4NF feltételt, akkor $X \twoheadrightarrow Y$ is megsérti.
Ha nincs ilyen, akkor tovább kell keresnünk. (Létezik erre is algoritmus, de azt most nem ismertetjük.)

Felbontás 4NF-re

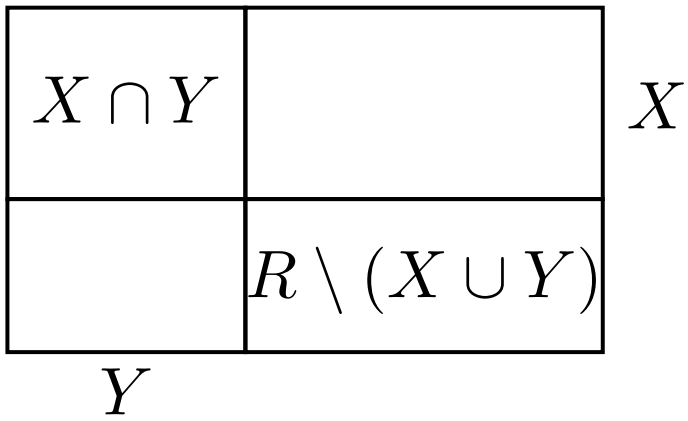
Tétel

Legyen (R, F) egy séma, ahol F funkcionális és többértékű függések halmaza. Ekkor (R, F) felbontható hűségesen 4NF relációkra.

Algoritmus: Hasonlóan BCNF-hez, mindig két valódi részre bontjuk hűségesen, addig, amíg mindegyik rész 4NF nem lesz.
Keresünk egy $X \twoheadrightarrow Y$ függést, ami megsérti a 4NF feltételt.
Ha van $X \rightarrow Y$, ami megsérti a 4NF feltételt, akkor $X \twoheadrightarrow Y$ is megsérti.
Ha nincs ilyen, akkor tovább kell keresnünk. (Létezik erre is algoritmus, de azt most nem ismertetjük.)

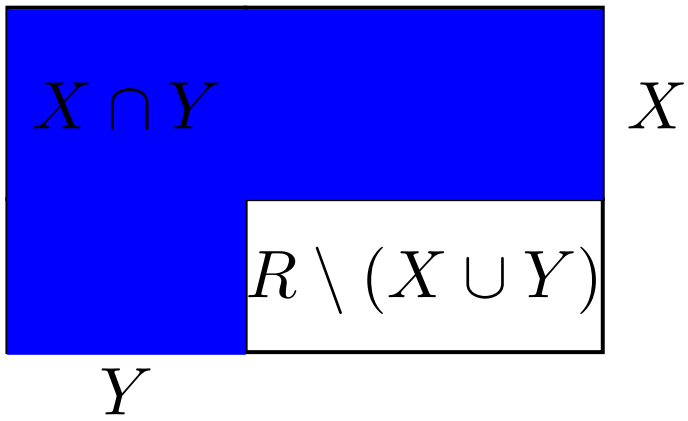
Felbontás 4NF-re

$R_1 = XY$ $\quad$ $R_2 = R \setminus (Y \setminus X)$ $(= X \cup (R \setminus Y))$



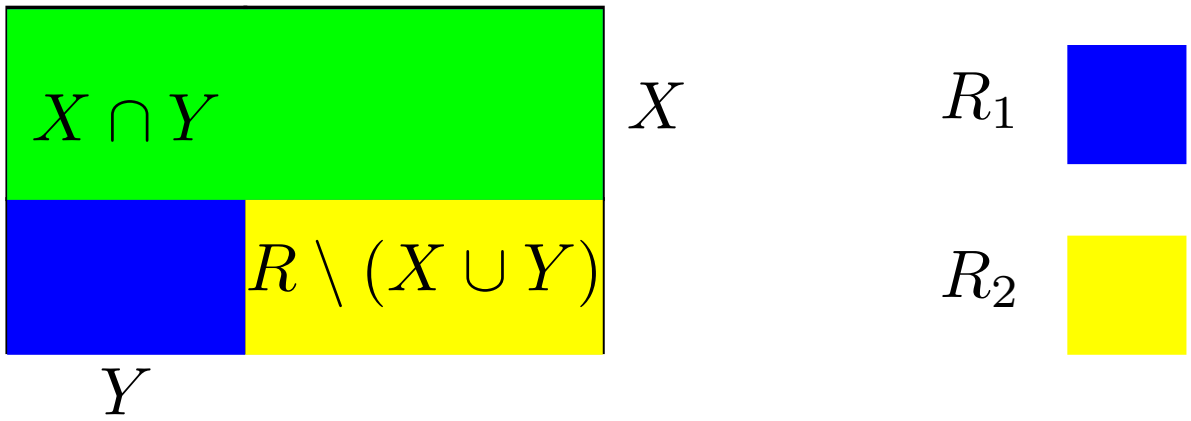
Felbontás 4NF-re

$R_1 = XY$ $\quad$ $R_2 = R \setminus (Y \setminus X)$ $\ (= X \cup (R \setminus Y))$



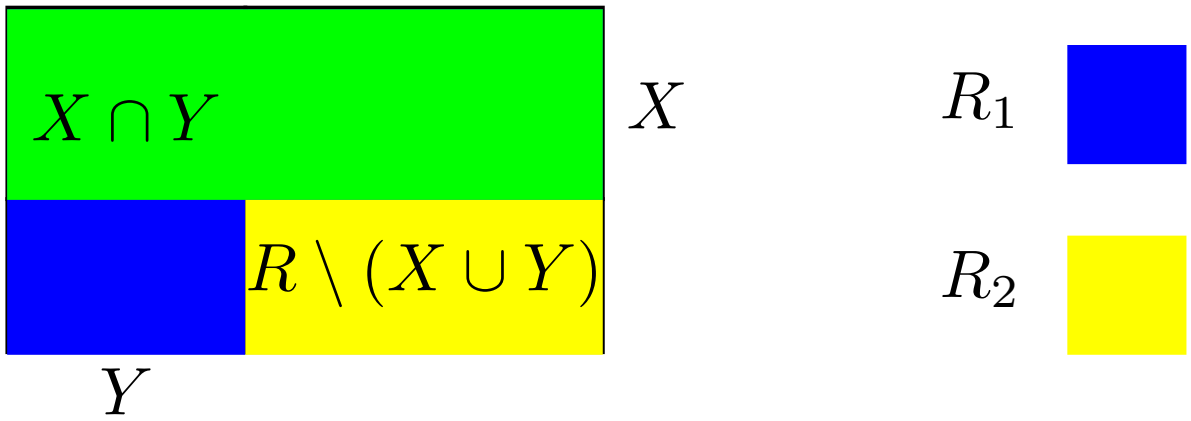
Felbontás 4NF-re

$R_1 = XY$ $\quad$ $R_2 = R \setminus (Y \setminus X)$ $(= X \cup (R \setminus Y))$



Felbontás 4NF-re

$R_1 = XY$ $\quad$ $R_2 = R \setminus (Y \setminus X)$ $(= X \cup (R \setminus Y))$

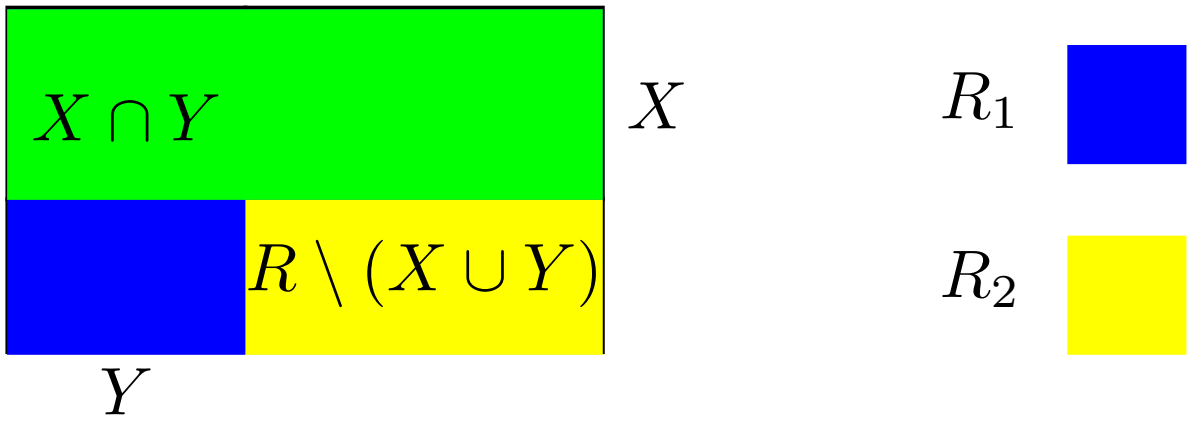


Ez valódi felbontás:

Ha $R_1 = R \implies X \twoheadrightarrow Y$ triviális függés lenne, ↯ .

Felbontás 4NF-re

$R_1 = XY$ $\quad$ $R_2 = R \setminus (Y \setminus X)$ $(= X \cup (R \setminus Y))$



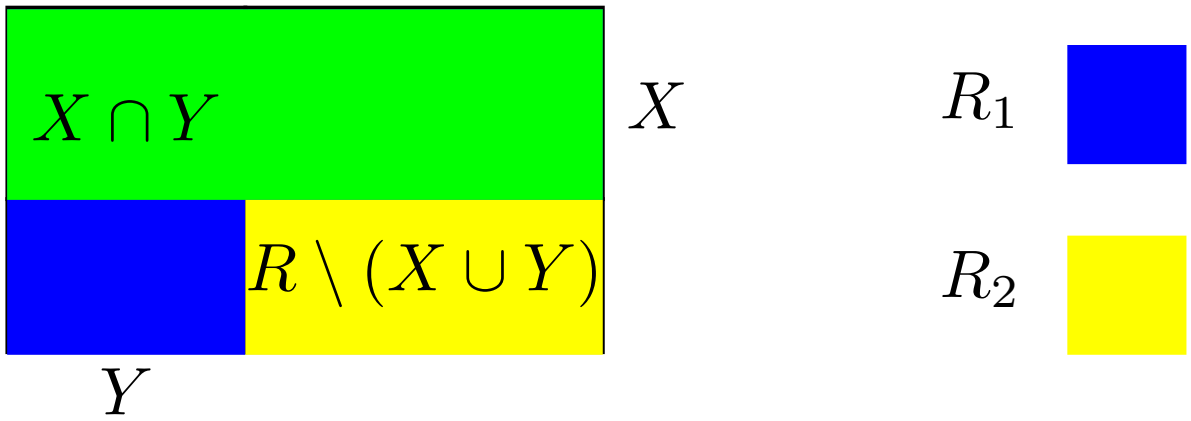
Ez valódi felbontás:

Ha $R_1 = R \implies X \twoheadrightarrow Y$ triviális függés lenne, ↯.

Ha $R_2 = R \implies Y \subseteq X \implies X \twoheadrightarrow Y$ triviális függés lenne, ↯.

Felbontás 4NF-re

$R_1 = XY$ $\quad$ $R_2 = R \setminus (Y \setminus X)$ $(= X \cup (R \setminus Y))$



Ez valódi felbontás:

Ha $R_1 = R \implies X \twoheadrightarrow Y$ triviális függés lenne, ↯.

Ha $R_2 = R \implies Y \subseteq X \implies X \twoheadrightarrow Y$ triviális függés lenne, ↯.

Ez hűséges felbontás:

$R_1 \cap R_2 = X$; $R_2 \setminus R_1 = R \setminus XY$ és $X \twoheadrightarrow R \setminus XY$ fennáll $X \twoheadrightarrow Y$ miatt.

Példa

R(Színész, Város, Utca, Filmcím, Filmév)
$F = \{\text{Színész} \twoheadrightarrow \text{Város, Utca}\}$

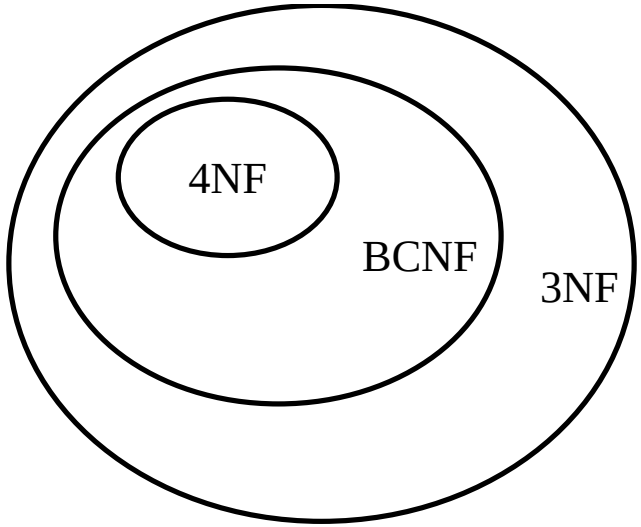
Példa

R(Színész, Város, Utca, Filmcím, Filmév)
$F = \{\text{Színész} \twoheadrightarrow \text{Város, Utca}\}$
Ez megsérti a 4NF tulajdonságot, ha Színész nem szuperkulcs.

Példa

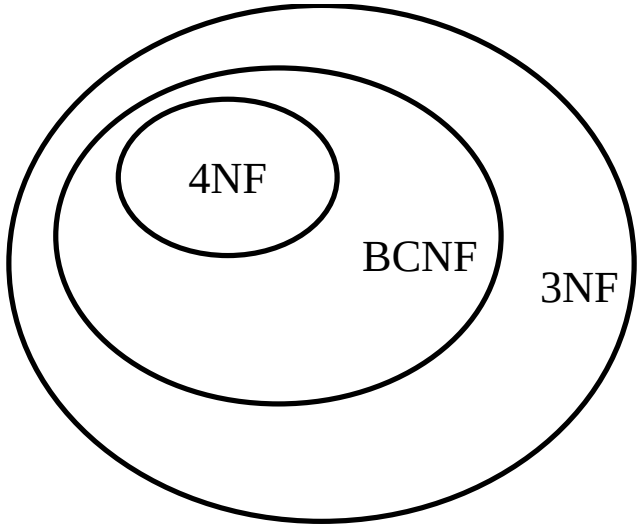
R(Színész, Város, Utca, Filmcím, Filmév)
$F = \{\text{Színész} \twoheadrightarrow \text{Város, Utca}\}$
Ez megsérti a 4NF tulajdonságot, ha Színész nem szuperkulcs.
4NF felbontás: R_1 =(Színész, Város, Utca) R_2 =(Színész, Filmcím, Filmév)

Normálformák összefoglalása



Jellemzők	3NF	BCNF	4NF
Megszünteti a funkcionális függőségekből eredő redundanciát	Gyakran	Igen	Igen
Megszünteti a többértékű függőségekből eredő redundanciát	Nem	Nem	Igen
Az ilyen felbontás megőrzi a funkcionális függőségeket	Igen	Lehet	Lehet
Az ilyen felbontás megőrzi a többértékű függőségeket	Lehet	Lehet	Lehet

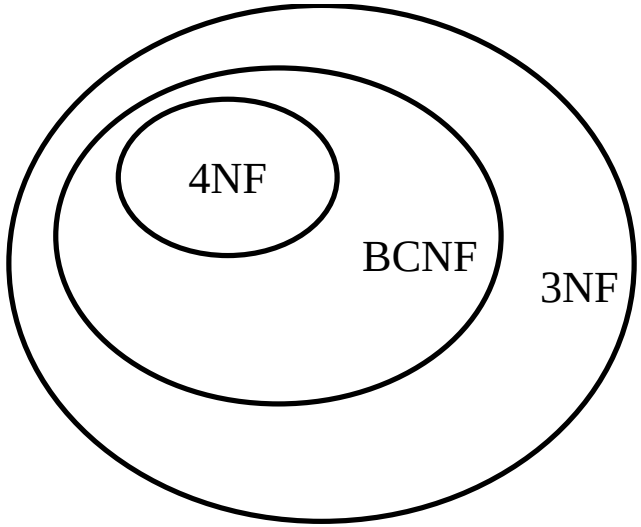
Normálformák összefoglalása



Jellemzők	3NF	BCNF	4NF
Megszünteti a funkcionális függőségekből eredő redundanciát	Gyakran	Igen	Igen
Megszünteti a többértékű függőségekből eredő redundanciát	Nem	Nem	Igen
Az ilyen felbontás megőrzi a funkcionális függőségeket	Igen	Lehet	Lehet
Az ilyen felbontás megőrzi a többértékű függőségeket	Lehet	Lehet	Lehet

Fontos elv: Ne bontsuk tovább, amit már nem muszáj.

Normálformák összefoglalása

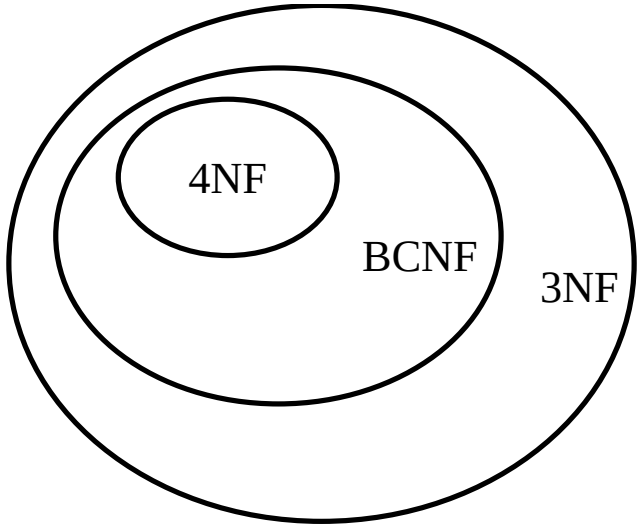


Jellemzők	3NF	BCNF	4NF
Megszünteti a funkcionális függőségekből eredő redundanciát	Gyakran	Igen	Igen
Megszünteti a többértékű függőségekből eredő redundanciát	Nem	Nem	Igen
Az ilyen felbontás megőrzi a funkcionális függőségeket	Igen	Lehet	Lehet
Az ilyen felbontás megőrzi a többértékű függőségeket	Lehet	Lehet	Lehet

Fontos elv: Ne bontsuk tovább, amit már nem muszáj.

A normalizálás azért fontos, mert . . .

Normálformák összefoglalása



Jellemzők	3NF	BCNF	4NF
Megszünteti a funkcionális függőségekből eredő redundanciát	Gyakran	Igen	Igen
Megszünteti a többértékű függőségekből eredő redundanciát	Nem	Nem	Igen
Az ilyen felbontás megőrzi a funkcionális függőségeket	Igen	Lehet	Lehet
Az ilyen felbontás megőrzi a többértékű függőségeket	Lehet	Lehet	Lehet

Fontos elv: Ne bontsuk tovább, amit már nem muszáj.

A normalizálás azért fontos, mert . . .

Adatbázisok elmélete

Adatbázisrendszerek megvalósítása, fizikai szervezés

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

17. előadás

Adatbázisrendszerek megvalósítása

Eddig az adatbáziskezelők működéséről tanultunk. Az év hátralevő részében az ilyen rendszerek belső működését tanulmányozzuk egy kicsit.

Adatbázisrendszerek megvalósítása

Eddig az adatbáziskezelők működéséről tanultunk. Az év hátralevő részében az ilyen rendszerek belső működését tanulmányozzuk egy kicsit.
Három nagyobb témakör:

1. Lekérdezésfeldolgozás: hogyan értékelődnek ki a lekérdezések, milyen módszerek vannak a lekérdezések végrehajtására?

Adatbázisrendszerek megvalósítása

Eddig az adatbáziskezelők működéséről tanultunk. Az év hátralevő részében az ilyen rendszerek belső működését tanulmányozzuk egy kicsit.
Három nagyobb témakör:

1. Lekérdezésfeldolgozás: hogyan értékelődnek ki a lekérdezések, milyen módszerek vannak a lekérdezések végrehajtására?
2. Fizikai szervezés, tárkezelés: hogyan tároljuk a relációkat oly módon, hogy gyorsan lehessen keresni, illetve módosítani?

Adatbázisrendszerek megvalósítása

Eddig az adatbáziskezelők működéséről tanultunk. Az év hátralevő részében az ilyen rendszerek belső működését tanulmányozzuk egy kicsit.
Három nagyobb témakör:

1. Lekérdezésfeldolgozás: hogyan értékelődnek ki a lekérdezések, milyen módszerek vannak a lekérdezések végrehajtására?
2. Fizikai szervezés, tárkezelés: hogyan tároljuk a relációkat oly módon, hogy gyorsan lehessen keresni, illetve módosítani?
3. Tranzakciókezelés: többfelhasználós működés biztosítása, illetve rendszerhibák elleni védelem.

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*
- típusellenőrzések $\Longrightarrow$ *pl. LIKE használatakor csak karakterlánc lehet*

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*
- típusellenőrzések $\Longrightarrow$ *pl. LIKE használatakor csak karakterlánc lehet*

Logikai lekérdezési terv:

- Átírás (kibővített) relációs algebrai alakra

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*
- típusellenőrzések $\Longrightarrow$ *pl. LIKE használatakor csak karakterlánc lehet*

Logikai lekérdezési terv:

- Átírás (kibővített) relációs algebrai alakra
- Transzformációk $\Longrightarrow$ *több terv, gyorsítás*

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*
- típusellenőrzések $\Longrightarrow$ *pl. LIKE használatakor csak karakterlánc lehet*

Logikai lekérdezési terv:

- Átírás (kibővített) relációs algebrai alakra
- Transzformációk $\Longrightarrow$ *több terv, gyorsítás*
- Legjobb terv kiválasztása költségbecsléssel

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*
- típusellenőrzések $\Longrightarrow$ *pl. LIKE használatakor csak karakterlánc lehet*

Logikai lekérdezési terv:

- Átírás (kibővített) relációs algebrai alakra
- Transzformációk $\Longrightarrow$ *több terv, gyorsítás*
- Legjobb terv kiválasztása költségbecsléssel

Fizikai terv kiválasztása:

- Algoritmusok a műveletekhez

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*
- típusellenőrzések $\Longrightarrow$ *pl. LIKE használatakor csak karakterlánc lehet*

Logikai lekérdezési terv:

- Átírás (kibővített) relációs algebrai alakra
- Transzformációk $\Longrightarrow$ *több terv, gyorsítás*
- Legjobb terv kiválasztása költségbecsléssel

Fizikai terv kiválasztása:

- Algoritmusok a műveletekhez
- Pufferkezelés

Lekérdezések végrehajtása, „optimalizálása”

Elemzés (parsing):

- szintaktikai ellenőrzés $\Longrightarrow$ *megfelelő parancsok, megfelelő sorrendben*
- átírás elemzőfa alakra

Előfeldolgozó:

- Relációk használatának ellenőrzése $\Longrightarrow$ *van-e ilyen*
- Attribútumnevek használatának ellenőrzése $\Longrightarrow$ *pl. egyértelmű-e, melyik attribútum melyik relációban van, benne van-e egyáltalán*
- típusellenőrzések $\Longrightarrow$ *pl. LIKE használatakor csak karakterlánc lehet*

Logikai lekérdezési terv:

- Átírás (kibővített) relációs algebrai alakra
- Transzformációk $\Longrightarrow$ *több terv, gyorsítás*
- Legjobb terv kiválasztása költségbecsléssel

Fizikai terv kiválasztása:

- Algoritmusok a műveletekhez
- Pufferkezelés
- Közbülső relációk eltárolása

A relációs algebra kibővítése

Az SQL többet tud, mint a relációs algebra, de az extra dolgokat is át akarjuk írni relációs formába. Néhány különbség:

- Multihalmazok $\Longrightarrow \cap_H, \cap_M$

A relációs algebra kibővítése

Az SQL többet tud, mint a relációs algebra, de az extra dolgokat is át akarjuk írni relációs formába. Néhány különbség:

- Multihalmazok $\Longrightarrow \cap_H, \cap_M$
- Kiválasztásnál, $\overset{\bowtie}{\theta}$-nál a feltételben használhatunk aritmetikai műveleteket $\Longrightarrow \sigma_{A+B<5}(R),\ R \underset{A+R.B<C+S.B}{\bowtie} S$

A relációs algebra kibővítése

Az SQL többet tud, mint a relációs algebra, de az extra dolgokat is át akarjuk írni relációs formába. Néhány különbség:

- Multihalmazok $\Longrightarrow \cap_H, \cap_M$
- Kiválasztásnál, $\overset{\bowtie}{\theta}$-nál a feltételben használhatunk aritmetikai műveleteket $\Longrightarrow \sigma_{A+B<5}(R),\ R \overset{\bowtie}{_{A+R.B<C+S.B}} S$
- Vetítés aritmetikai műveletekkel és átnevezéssel $\Longrightarrow \pi_{A,\, B+C \to X}(R)$

A relációs algebra kibővítése

Az SQL többet tud, mint a relációs algebra, de az extra dolgokat is át akarjuk írni relációs formába. Néhány különbség:

- Multihalmazok $\Longrightarrow \cap_H, \cap_M$
- Kiválasztásnál, $\overset{\bowtie}{\theta}$-nál a feltételben használhatunk aritmetikai műveleteket $\Longrightarrow \sigma_{A+B<5}(R),\ R \overset{\bowtie}{_{A+R.B<C+S.B}} S$
- Vetítés aritmetikai műveletekkel és átnevezéssel $\Longrightarrow \pi_{A,\, B+C \to X}(R)$
- Ismétlődések kiszűrése $\Longrightarrow \delta(R)$

A relációs algebra kibővítése

Az SQL többet tud, mint a relációs algebra, de az extra dolgokat is át akarjuk írni relációs formába. Néhány különbség:

- Multihalmazok $\Longrightarrow \cap_H, \cap_M$
- Kiválasztásnál, $\overset{\bowtie}{\theta}$-nál a feltételben használhatunk aritmetikai műveleteket $\Longrightarrow \sigma_{A+B<5}(R)$, $R \overset{\bowtie}{_{A+R.B<C+S.B}} S$
- Vetítés aritmetikai műveletekkel és átnevezéssel $\Longrightarrow \pi_{A,\, B+C \to X}(R)$
- Ismétlődések kiszűrése $\Longrightarrow \delta(R)$
- Csoportosítások, aggregátumok $\Longrightarrow$ SELECT A, $MIN(B)$ AS $minB$ FROM R GROUP BY A $\Longrightarrow \gamma_{A,MIN(B)\to minB}(R)$

Fizikai végrehajtás

Leginkább az I/O műveletigény érdekes. Ha „túl nagy” a számítási igény az is baj lehet.

Soronkénti, unáris műveletek: Kiválasztás és vetítés. Egyszerre csak egy sort kell vizsgálni, az algoritmus nem függ a memória nagyságától.

Fizikai végrehajtás

Leginkább az I/O műveletigény érdekes. Ha „túl nagy” a számítási igény az is baj lehet.

Soronkénti, unáris műveletek: Kiválasztás és vetítés. Egyszerre csak egy sort kell vizsgálni, az algoritmus nem függ a memória nagyságától.

Unáris, teljes relációs műveletek: Pl. $\delta(R), \gamma(R)$. Ha nem fér el a reláció a memóriában, akkor mást kell csinálni.

Fizikai végrehajtás

Leginkább az I/O műveletigény érdekes. Ha „túl nagy” a számítási igény az is baj lehet.

Soronkénti, unáris műveletek: Kiválasztás és vetítés. Egyszerre csak egy sort kell vizsgálni, az algoritmus nem függ a memória nagyságától.

Unáris, teljes relációs műveletek: Pl. $\delta(R), \gamma(R)$. Ha nem fér el a reláció a memóriában, akkor mást kell csinálni.

Bináris, teljes relációs műveletek: $\cup, \cap, \setminus, \times, \bowtie$. Sok minden függ a méretektől.

Jelölés: Az adatokat a külső tárról blokkonként olvassuk be. Az R reláció tárolásához szükséges blokkok számát $B(R)$-rel jelöljük.

Fizikai végrehajtás

Leginkább az I/O műveletigény érdekes. Ha „túl nagy” a számítási igény az is baj lehet.

Soronkénti, unáris műveletek: Kiválasztás és vetítés. Egyszerre csak egy sort kell vizsgálni, az algoritmus nem függ a memória nagyságától.

Unáris, teljes relációs műveletek: Pl. $\delta(R), \gamma(R)$. Ha nem fér el a reláció a memóriában, akkor mást kell csinálni.

Bináris, teljes relációs műveletek: $\cup, \cap, \setminus, \times, \bowtie$. Sok minden függ a méretektől.

Jelölés: Az adatokat a külső tárról blokkonként olvassuk be. Az R reláció tárolásához szükséges blokkok számát $B(R)$-rel jelöljük.
A bels? memória mérete szintén blokkokban mérve legyen M.

Fizikai végrehajtás

Leginkább az I/O műveletigény érdekes. Ha „túl nagy” a számítási igény az is baj lehet.

Soronkénti, unáris műveletek: Kiválasztás és vetítés. Egyszerre csak egy sort kell vizsgálni, az algoritmus nem függ a memória nagyságától.

Unáris, teljes relációs műveletek: Pl. $\delta(R), \gamma(R)$. Ha nem fér el a reláció a memóriában, akkor mást kell csinálni.

Bináris, teljes relációs műveletek: $\cup, \cap, \setminus, \times, \bowtie$. Sok minden függ a méretektől.

Jelölés: Az adatokat a külső tárról blokkonként olvassuk be. Az R reláció tárolásához szükséges blokkok számát $B(R)$-rel jelöljük.
A bels? memória mérete szintén blokkokban mérve legyen M.
$\sigma_C(R)$ **végrehajtása:** Blokkonként beolvassuk R-et. Soronként megnézzük teljesül-e C. Ha igen, kiírjuk.

Fizikai végrehajtás

Leginkább az I/O műveletigény érdekes. Ha „túl nagy” a számítási igény az is baj lehet.

Soronkénti, unáris műveletek: Kiválasztás és vetítés. Egyszerre csak egy sort kell vizsgálni, az algoritmus nem függ a memória nagyságától.

Unáris, teljes relációs műveletek: Pl. $\delta(R), \gamma(R)$. Ha nem fér el a reláció a memóriában, akkor mást kell csinálni.

Bináris, teljes relációs műveletek: $\cup, \cap, \setminus, \times, \bowtie$. Sok minden függ a méretektől.

Jelölés: Az adatokat a külső tárról blokkonként olvassuk be. Az R reláció tárolásához szükséges blokkok számát $B(R)$-rel jelöljük.
A bels? memória mérete szintén blokkokban mérve legyen M.
$\sigma_C(R)$ **végrehajtása:** Blokkonként beolvassuk R-et. Soronként megnézzük teljesül-e C. Ha igen, kiírjuk.
I/O műveletigény: $B(R)$

Fizikai végrehajtás

Leginkább az I/O műveletigény érdekes. Ha „túl nagy” a számítási igény az is baj lehet.

Soronkénti, unáris műveletek: Kiválasztás és vetítés. Egyszerre csak egy sort kell vizsgálni, az algoritmus nem függ a memória nagyságától.

Unáris, teljes relációs műveletek: Pl. $\delta(R), \gamma(R)$. Ha nem fér el a reláció a memóriában, akkor mást kell csinálni.

Bináris, teljes relációs műveletek: $\cup, \cap, \setminus, \times, \bowtie$. Sok minden függ a méretektől.

Jelölés: Az adatokat a külső tárról blokkonként olvassuk be. Az R reláció tárolásához szükséges blokkok számát $B(R)$-rel jelöljük.
A bels? memória mérete szintén blokkokban mérve legyen M.
$\sigma_C(R)$ **végrehajtása:** Blokkonként beolvassuk R-et. Soronként megnézzük teljesül-e C. Ha igen, kiírjuk.
I/O műveletigény: $B(R)$
Ha $\sigma_{A='c'}(R)$-t akarjuk, és van index A-ra: sokkal gyorsabb lehet.

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ végrehajtása:

- *Ha $B(S) < M - 1$, azaz S belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

  I/O műveletigény: $B(S) + B(R)$

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

  I/O műveletigény: $B(S) + B(R)$
- *Ha* $B(R) > B(S) > M - 1$*:* beágyazott ciklusú algoritmus

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

  I/O műveletigény: $B(S) + B(R)$
- *Ha* $B(R) > B(S) > M - 1$*:* beágyazott ciklusú algoritmus
  Beolvasunk minél több blokkot a memóriába S-ből, utána ugyanazt csináljuk mint fenn.

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

  I/O műveletigény: $B(S) + B(R)$
- *Ha* $B(R) > B(S) > M - 1$*:* beágyazott ciklusú algoritmus
  Beolvasunk minél több blokkot a memóriába S-ből, utána ugyanazt csináljuk mint fenn.
  I/O műveletigény: $B(S) + B(S)B(R)/(M - 1) \approx B(S)B(R)/M$

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

  I/O műveletigény: $B(S) + B(R)$
- *Ha* $B(R) > B(S) > M - 1$*:* beágyazott ciklusú algoritmus
  Beolvasunk minél több blokkot a memóriába S-ből, utána ugyanazt csináljuk mint fenn.
  I/O műveletigény: $B(S) + B(S)B(R)/(M - 1) \approx B(S)B(R)/M$
- *Ha* $B(R), B(S) \leq M^2$*:* rendezéses algoritmus

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

  I/O műveletigény: $B(S) + B(R)$
- *Ha* $B(R) > B(S) > M - 1$*:* beágyazott ciklusú algoritmus
  Beolvasunk minél több blokkot a memóriába S-ből, utána ugyanazt csináljuk mint fenn.
  I/O műveletigény: $B(S) + B(S)B(R)/(M - 1) \approx B(S)B(R)/M$
- *Ha* $B(R), B(S) \leq M^2$*:* rendezéses algoritmus
  Y kulcs szerint rendezzük R-et és S-et összefésüléses rendezéssel. Vesszük az összes y kulcsú sort a két lista elejéről és kiírjuk az összes párt. (Feltettük, hogy az összes y kulcsú sor elfér a memóriában.)

Fizikai végrehajtás

$R(X, Y) \bowtie S(Y, Z)$ **végrehajtása:**

- *Ha* $B(S) < M - 1$*, azaz* S *belefér a memóriába:* egymenetes algoritmus
  1. Beolvassuk S-et és hashtáblát vagy B-fát készítünk, ahol a kulcs Y attribútumai.
  2. Beolvasunk egy blokkot R-ből. Minden sorára kikeressük a passzoló S-beli sorokat. Az eredményt kiírjuk.

  I/O műveletigény: $B(S) + B(R)$
- *Ha* $B(R) > B(S) > M - 1$*:* beágyazott ciklusú algoritmus
  Beolvasunk minél több blokkot a memóriába S-ből, utána ugyanazt csináljuk mint fenn.
  I/O műveletigény: $B(S) + B(S)B(R)/(M - 1) \approx B(S)B(R)/M$
- *Ha* $B(R), B(S) \leq M^2$*:* rendezéses algoritmus
  Y kulcs szerint rendezzük R-et és S-et összefésüléses rendezéssel. Vesszük az összes y kulcsú sort a két lista elejéről és kiírjuk az összes párt. (Feltettük, hogy az összes y kulcsú sor elfér a memóriában.)
  I/O műveletigény: $5(B(S) + B(R))$

Fizikai végrehajtás

- *Ha* $\min(B(R), B(S)) \leq M^2$*:* hasheléses algoritmus

Fizikai végrehajtás

- *Ha* $\min(B(R), B(S)) \leq M^2$*:* hasheléses algoritmus
  Y kulcs szerint vödrös hashelést végzünk R-re és S-re. (Ha közben megtelik egy vödör, azt kiírjuk.) A kapott R_i, S_i vödrökkel egymenetes algoritmust végzünk.

Fizikai végrehajtás

- *Ha* $\min(B(R), B(S)) \leq M^2$*:* hasheléses algoritmus
  Y kulcs szerint vödrös hashelést végzünk R-re és S-re. (Ha közben megtelik egy vödör, azt kiírjuk.) A kapott R_i, S_i vödrökkel egymenetes algoritmust végzünk.
  I/O műveletigény: $3(B(S) + B(R))$

Fizikai végrehajtás

- *Ha* $\min(B(R), B(S)) \leq M^2$*:* hasheléses algoritmus
  Y kulcs szerint vödrös hashelést végzünk R-re és S-re. (Ha közben megtelik egy vödör, azt kiírjuk.) A kapott R_i, S_i vödrökkel egymenetes algoritmust végzünk.
  I/O műveletigény: $3(B(S) + B(R))$
- *Ha van index S-re Y szerint:* indexet használó algoritmus

Fizikai végrehajtás

- *Ha* $\min(B(R), B(S)) \leq M^2$*:* hasheléses algoritmus
  Y kulcs szerint vödrös hashelést végzünk R-re és S-re. (Ha közben megtelik egy vödör, azt kiírjuk.) A kapott R_i, S_i vödrökkel egymenetes algoritmust végzünk.
  I/O műveletigény: $3(B(S) + B(R))$
- *Ha van index S-re Y szerint:* indexet használó algoritmus
  R-et blokkonként olvassuk be, az index alapján keressük ki a hozzá passzoló sorokat.

Fizikai végrehajtás

- *Ha* $\min(B(R), B(S)) \leq M^2$*:* hasheléses algoritmus
  Y kulcs szerint vödrös hashelést végzünk R-re és S-re. (Ha közben megtelik egy vödör, azt kiírjuk.) A kapott R_i, S_i vödrökkel egymenetes algoritmust végzünk.
  I/O műveletigény: $3(B(S) + B(R))$
- *Ha van index S-re Y szerint:* indexet használó algoritmus
  R-et blokkonként olvassuk be, az index alapján keressük ki a hozzá passzoló sorokat.
  Átlagos I/O műveletigény: $B(S)B(R)/V(S, Y)$, ahol $V(S, Y)$: Y értékkészletének számossága S-ben.

Fizikai végrehajtás

- *Ha* $\min(B(R), B(S)) \leq M^2$*:* hasheléses algoritmus
  Y kulcs szerint vödrös hashelést végzünk R-re és S-re. (Ha közben megtelik egy vödör, azt kiírjuk.) A kapott R_i, S_i vödrökkel egymenetes algoritmust végzünk.
  I/O műveletigény: $3(B(S) + B(R))$
- *Ha van index S-re Y szerint:* indexet használó algoritmus
  R-et blokkonként olvassuk be, az index alapján keressük ki a hozzá passzoló sorokat.
  Átlagos I/O műveletigény: $B(S)B(R)/V(S, Y)$, ahol $V(S, Y)$: Y értékkészletének számossága S-ben.

A többi műveletet is hasonló ötletekkel lehet végrehajtani, azokat most nem részletezzük.

Adatbázisok elmélete

Lekérdezések optimalizálása

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

18. előadás

Optimalizálás

Triviális egyszerűsítések (főleg generált lekérdezések esetén hasznos):

Optimalizálás

Triviális egyszerűsítések (főleg generált lekérdezések esetén hasznos):

- $r \cap r = r$; $r \bowtie r = r$; $r \cup \emptyset = r$; $\sigma_C(\emptyset) = \emptyset$

Optimalizálás

Triviális egyszerűsítések (főleg generált lekérdezések esetén hasznos):

- $r \cap r = r$; $r \bowtie r = r$; $r \cup \emptyset = r$; $\sigma_C(\emptyset) = \emptyset$
- $\pi_X(r \cup s) = \pi_X(r) \cup \pi_X(s)$

Optimalizálás

Triviális egyszerűsítések (főleg generált lekérdezések esetén hasznos):

- $r \cap r = r$; $r \bowtie r = r$; $r \cup \emptyset = r$; $\sigma_C(\emptyset) = \emptyset$
- $\pi_X(r \cup s) = \pi_X(r) \cup \pi_X(s)$
- $\sigma_{A=B \wedge B=C \wedge A=C}(r) = \sigma_{A=B \wedge B=C}(r)$

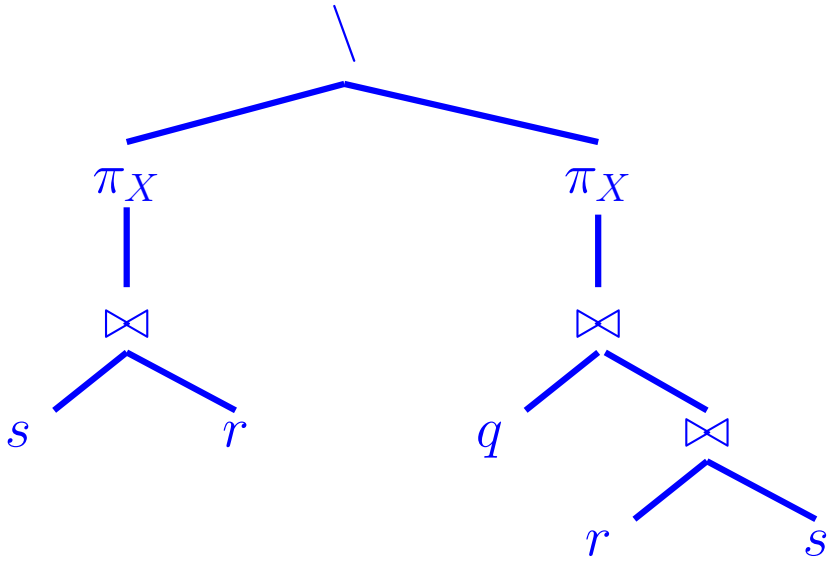
Optimalizálás

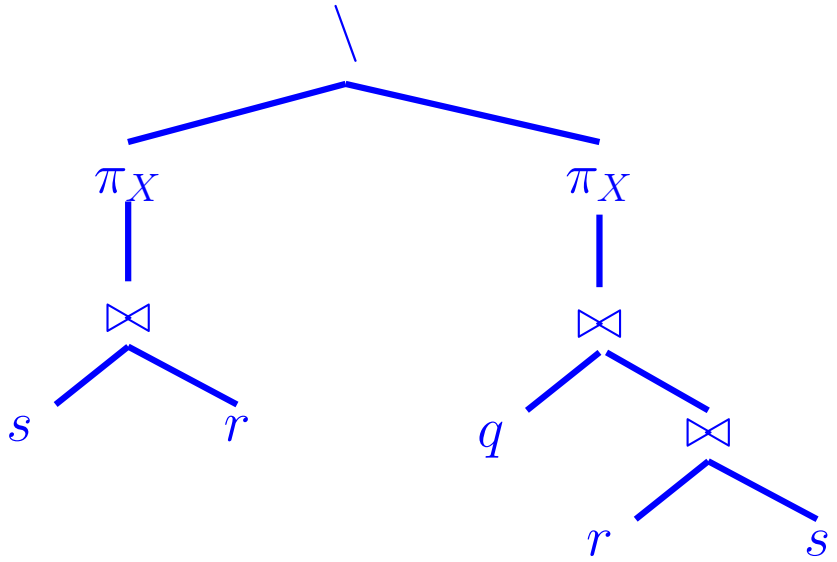
Ami többször előfordul, nem biztos, hogy érdemes mindig kiszámolni:

Optimalizálás

Ami többször előfordul, nem biztos, hogy érdemes mindig kiszámolni:

Pl. $\pi_X(s \bowtie r) \setminus \pi_X(q \bowtie r \bowtie s)$

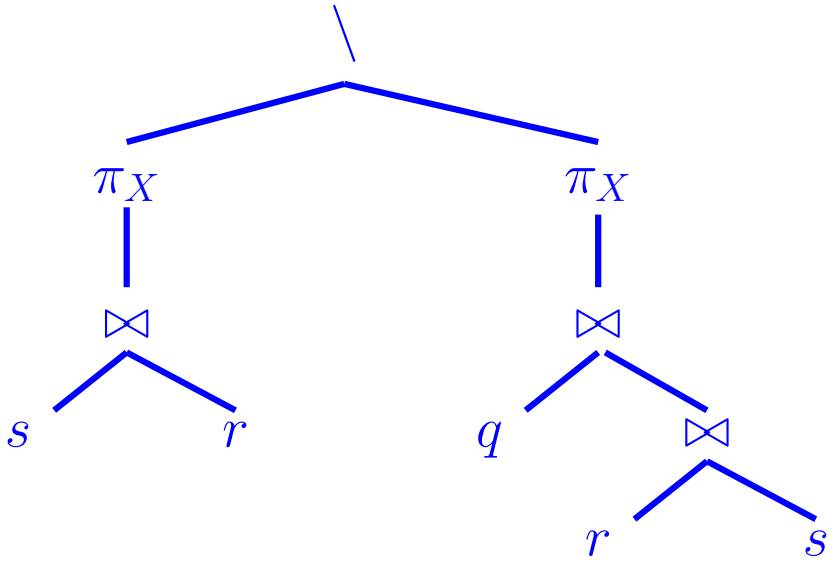


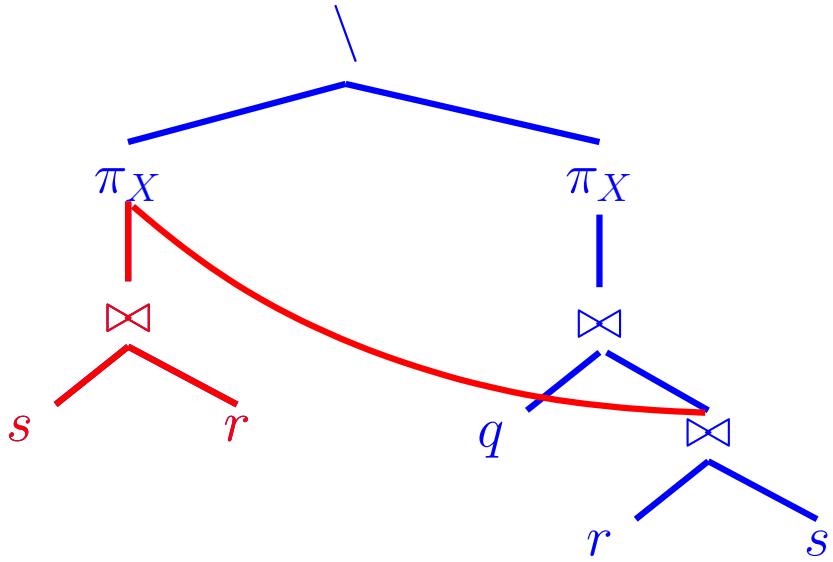


Optimalizálás

Ami többször előfordul, nem biztos, hogy érdemes mindig kiszámolni:

Pl. $\pi_X(s \bowtie r) \setminus \pi_X(q \bowtie r \bowtie s)$

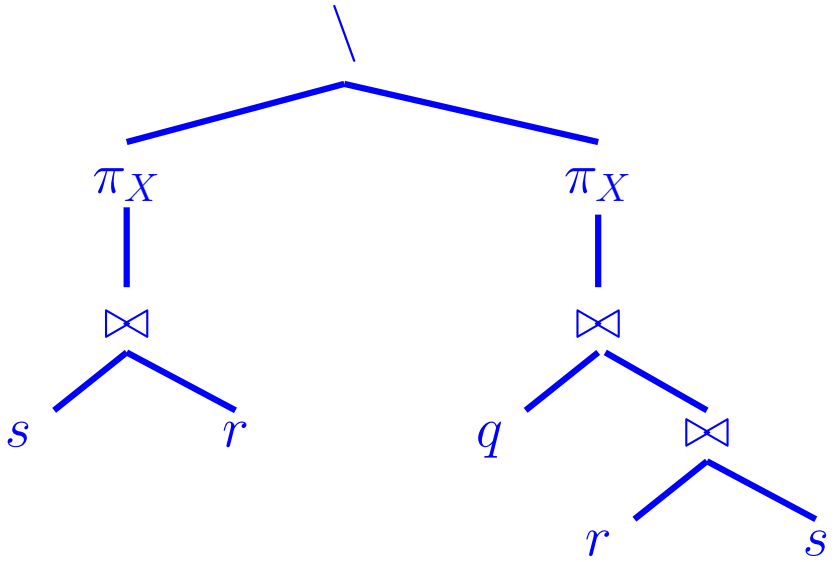


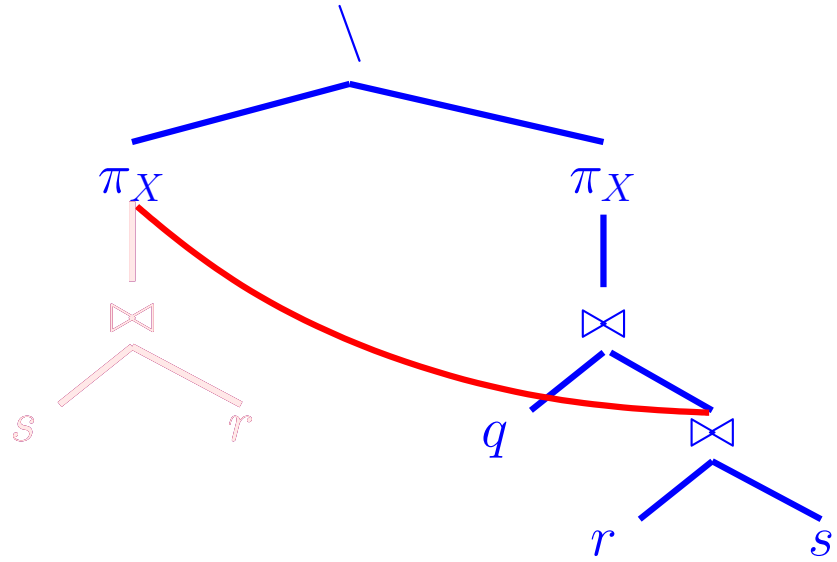


Optimalizálás

Ami többször előfordul, nem biztos, hogy érdemes mindig kiszámolni:

Pl. $\pi_X(s \bowtie r) \setminus \pi_X(q \bowtie r \bowtie s)$

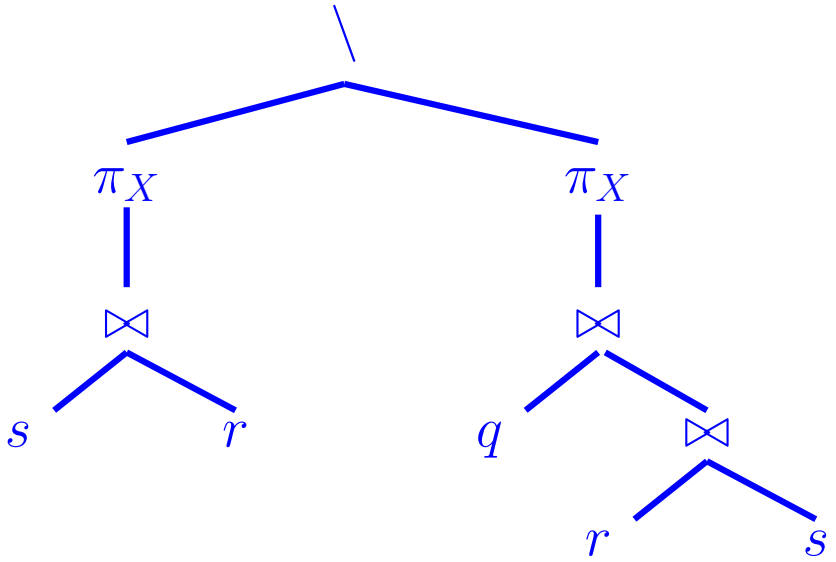


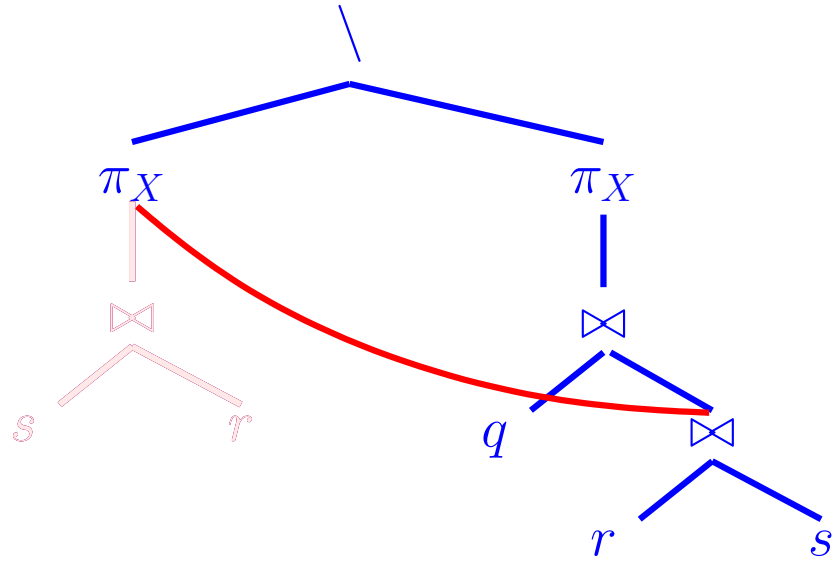


Optimalizálás

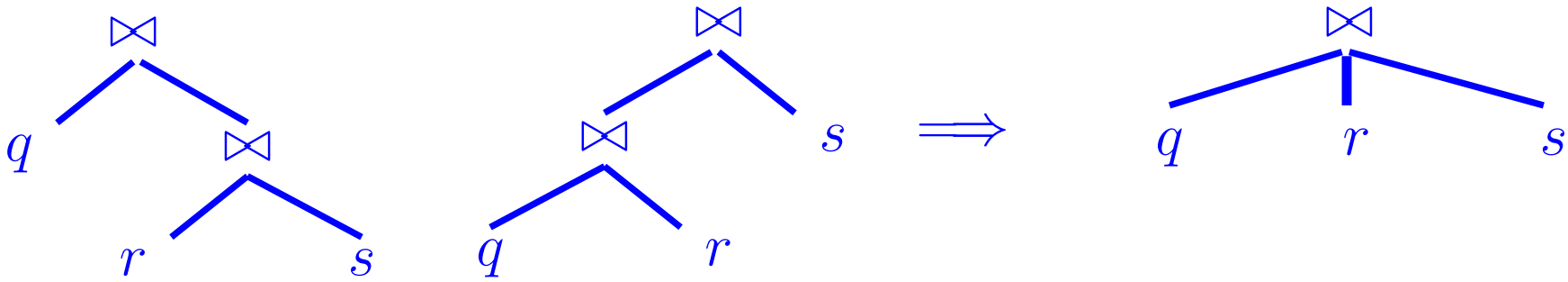
Ami többször előfordul, nem biztos, hogy érdemes mindig kiszámolni:

Pl. $\pi_X(s \bowtie r) \setminus \pi_X(q \bowtie r \bowtie s)$





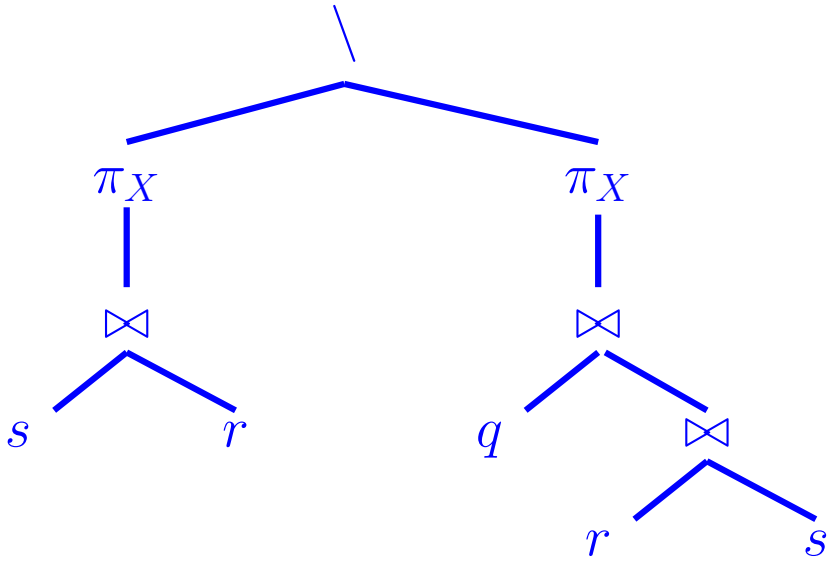
Asszociativitás kihasználható:

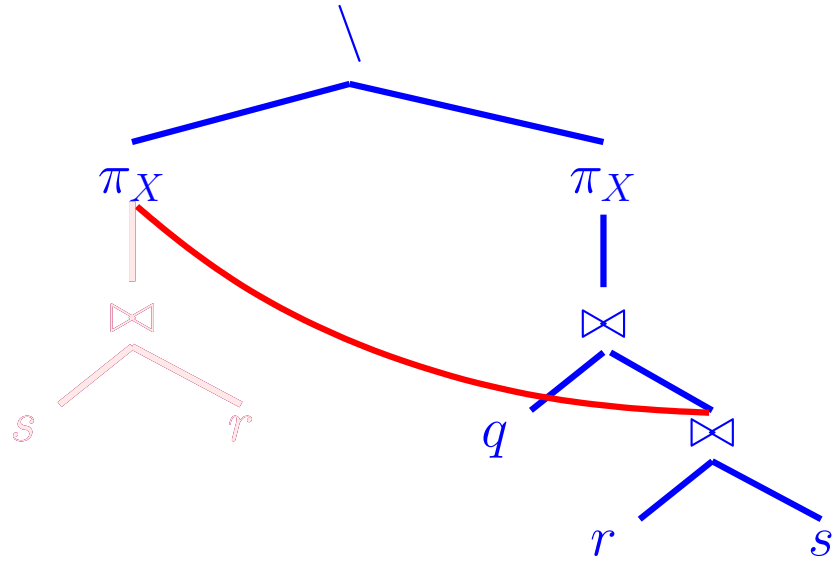


Optimalizálás

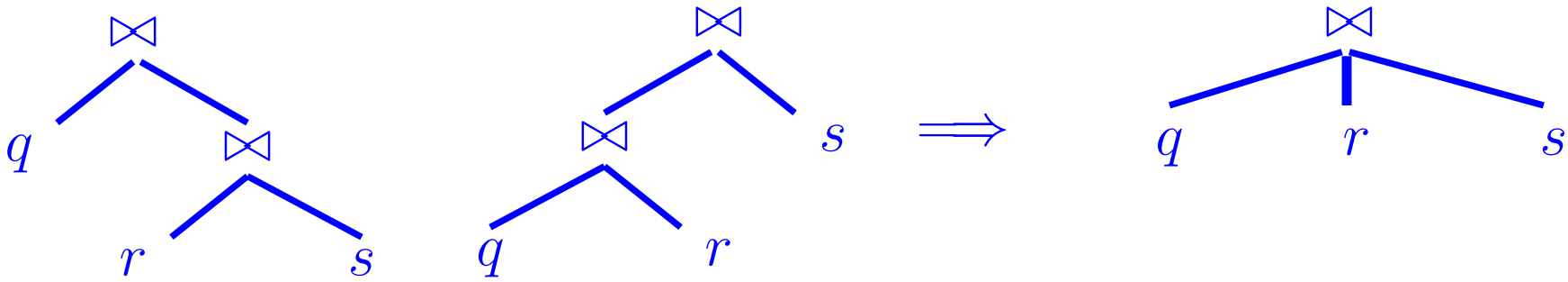
Ami többször előfordul, nem biztos, hogy érdemes mindig kiszámolni:

Pl. $\pi_X(s \bowtie r) \setminus \pi_X(q \bowtie r \bowtie s)$





Asszociativitás kihasználható:



Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A, B) \bowtie s(B, C) \bowtie q(C, D)$-t?

Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A, B) \bowtie s(B, C) \bowtie q(C, D)$**-t?**
Szélsőséges esetben lehet, hogy bár r, s, q mindegyikének 1000 sora van, de $r \bowtie s$-nek csak 1 sora, és $s \bowtie q$-nak 1000000 sora.

Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A, B) \bowtie s(B, C) \bowtie q(C, D)$-t?
Szélsőséges esetben lehet, hogy bár r, s, q mindegyikének 1000 sora van, de $r \bowtie s$-nek csak 1 sora, és $s \bowtie q$-nak 1000000 sora.
$\Longrightarrow$ Sokkal gyorsabb $(r(A, B) \bowtie s(B, C)) \bowtie q(C, D)$ kiszámolása.

Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A, B) \bowtie s(B, C) \bowtie q(C, D)$-t?
Szélsőséges esetben lehet, hogy bár r, s, q mindegyikének 1000 sora van, de $r \bowtie s$-nek csak 1 sora, és $s \bowtie q$-nak 1000000 sora.
$\Longrightarrow$ Sokkal gyorsabb $(r(A, B) \bowtie s(B, C)) \bowtie q(C, D)$ kiszámolása.
Ezt persze előre nem lehet tudni biztosan.

Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A,B) \bowtie s(B,C) \bowtie q(C,D)$-t?
Szélsőséges esetben lehet, hogy bár r, s, q mindegyikének 1000 sora van, de $r \bowtie s$-nek csak 1 sora, és $s \bowtie q$-nak 1000000 sora.
$\Longrightarrow$ Sokkal gyorsabb $(r(A,B) \bowtie s(B,C)) \bowtie q(C,D)$ kiszámolása.
Ezt persze előre nem lehet tudni biztosan. $\Longrightarrow$ Statisztikákat vezetünk a relációk attribútumaiban előforduló értékekről
Ebből lehet becsülni a költségeket + dinamikus programozás vagy mohó algoritmus.

Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A,B) \bowtie s(B,C) \bowtie q(C,D)$**-t?**
Szélsőséges esetben lehet, hogy bár r, s, q mindegyikének 1000 sora van, de $r \bowtie s$-nek csak 1 sora, és $s \bowtie q$-nak 1000000 sora.
$\Longrightarrow$ Sokkal gyorsabb $(r(A,B) \bowtie s(B,C)) \bowtie q(C,D)$ kiszámolása.
Ezt persze előre nem lehet tudni biztosan. $\Longrightarrow$ Statisztikákat vezetünk a relációk attribútumaiban előforduló értékekről
Ebből lehet becsülni a költségeket + dinamikus programozás vagy mohó algoritmus.

Igazi optimumot nehéz megtalálni:

Tétel

Annak eldöntése NP-teljes, hogy néhány reláció természetes illesztésének van-e legalább egy sora.

Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A,B) \bowtie s(B,C) \bowtie q(C,D)$-t?
Szélsőséges esetben lehet, hogy bár r, s, q mindegyikének 1000 sora van, de $r \bowtie s$-nek csak 1 sora, és $s \bowtie q$-nak 1000000 sora.
$\Longrightarrow$ Sokkal gyorsabb $(r(A,B) \bowtie s(B,C)) \bowtie q(C,D)$ kiszámolása.
Ezt persze előre nem lehet tudni biztosan. $\Longrightarrow$ Statisztikákat vezetünk a relációk attribútumaiban előforduló értékekről
Ebből lehet becsülni a költségeket + dinamikus programozás vagy mohó algoritmus.

Igazi optimumot nehéz megtalálni:

Tétel

Annak eldöntése NP-teljes, hogy néhány reláció természetes illesztésének van-e legalább egy sora.

Tétel

Az optimum megtalálása NP-nehéz probléma.

Bizonyítás

Tétel

Annak eldöntése NP-nehéz, hogy néhány reláció természetes illesztésének van-e legalább egy sora.

Bizonyítás.

Visszavezetjük rá a 3-SZÍN problémát. Adott egy gráf, kérdés színezhető-e 3 színnel.

Bizonyítás

Tétel

Annak eldöntése NP-nehéz, hogy néhány reláció természetes illesztésének van-e legalább egy sora.

Bizonyítás.

Visszavezetjük rá a 3-SZÍN problémát. Adott egy gráf, kérdés színezhető-e 3 színnel. A gráf minden e éléhez vegyünk fel egy-egy relációt.

Bizonyítás

Tétel

Annak eldöntése NP-nehéz, hogy néhány reláció természetes illesztésének van-e legalább egy sora.

Bizonyítás.

Visszavezetjük rá a 3-SZÍN problémát. Adott egy gráf, kérdés színezhető-e 3 színnel.
A gráf minden e éléhez vegyünk fel egy-egy relációt.
A reláció két attribútma az él két végpontja legyen, sorai pedig az összes lehetséges színpár. Például:

Bizonyítás

Tétel

Annak eldöntése NP-nehéz, hogy néhány reláció természetes illesztésének van-e legalább egy sora.

Bizonyítás.

Visszavezetjük rá a 3-SZÍN problémát. Adott egy gráf, kérdés színezhető-e 3 színnel.
A gráf minden e éléhez vegyünk fel egy-egy relációt.
A reláció két attribútma az él két végpontja legyen, sorai pedig az összes lehetséges színpár. Például:

$e = \{X, Y\}$

X	Y
piros	kék
piros	sárga
kék	piros
kék	sárga
sárga	piros
sárga	kék

$e' = \{X, Z\}$

X	Z
piros	kék
piros	sárga
kék	piros
kék	sárga
sárga	piros
sárga	kék

Bizonyítás

Tétel

Annak eldöntése NP-nehéz, hogy néhány reláció természetes illesztésének van-e legalább egy sora.

Bizonyítás.

Visszavezetjük rá a 3-SZÍN problémát. Adott egy gráf, kérdés színezhető-e 3 színnel.
A gráf minden e éléhez vegyünk fel egy-egy relációt.
A reláció két attribútma az él két végpontja legyen, sorai pedig az összes lehetséges színpár. Például:

$e = \{X, Y\}$

X	Y
piros	kék
piros	sárga
kék	piros
kék	sárga
sárga	piros
sárga	kék

$e' = \{X, Z\}$

X	Z
piros	kék
piros	sárga
kék	piros
kék	sárga
sárga	piros
sárga	kék

Bizonyítás

Bizonyítás.

Ha az összes élhez tartozó reláció természetes illesztésének van sora $\Longrightarrow$ egy sor minden csúcshoz rendel egy színt.

Bizonyítás

Bizonyítás.

Ha az összes élhez tartozó reláció természetes illesztésének van sora $\Longrightarrow$ egy sor minden csúcshoz rendel egy színt. Mivel az illesztés megfelel az egyes relációknak, egy él két végpontján nem lesz ugyanolyan szín.

Bizonyítás

Bizonyítás.

Ha az összes élhez tartozó reláció természetes illesztésének van sora $\Longrightarrow$ egy sor minden csúcshoz rendel egy színt. Mivel az illesztés megfelel az egyes relációknak, egy él két végpontján nem lesz ugyanolyan szín.
Ha van színezés

Bizonyítás

Bizonyítás.

Ha az összes élhez tartozó reláció természetes illesztésének van sora $\Longrightarrow$ egy sor minden csúcshoz rendel egy színt. Mivel az illesztés megfelel az egyes relációknak, egy él két végpontján nem lesz ugyanolyan szín.
Ha van színezés $\Longrightarrow$ a színezésben minden élre vegyük ki a megfelelő színpárt.

Bizonyítás

Bizonyítás.

Ha az összes élhez tartozó reláció természetes illesztésének van sora $\Longrightarrow$ egy sor minden csúcshoz rendel egy színt. Mivel az illesztés megfelel az egyes relációknak, egy él két végpontján nem lesz ugyanolyan szín.
Ha van színezés $\Longrightarrow$ a színezésben minden élre vegyük ki a megfelelő színpárt. Ezek a sorok összeillenek, lesz sor a természetes illesztésben. $\surd$ $\square$

Kiválasztás tologatása

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)

MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

Kiválasztás tologatása

ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)

MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right) \Longrightarrow$$

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG}\right) \bowtie \text{ÁRU}\right)$$

Kiválasztás tologatása

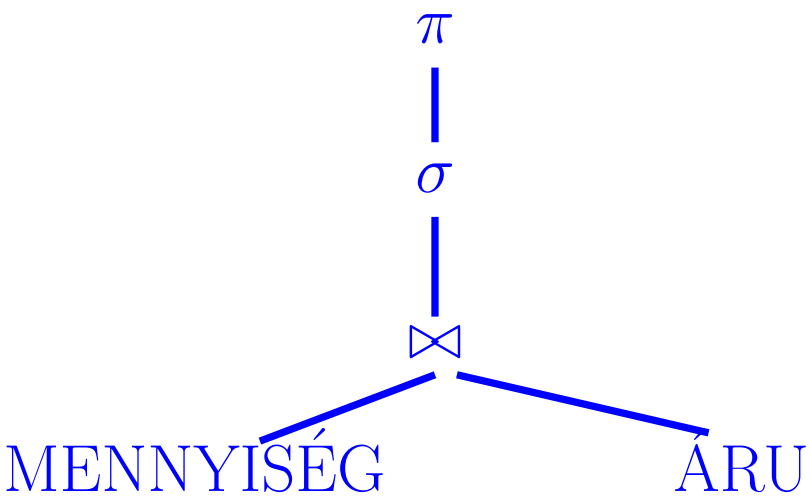
ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)

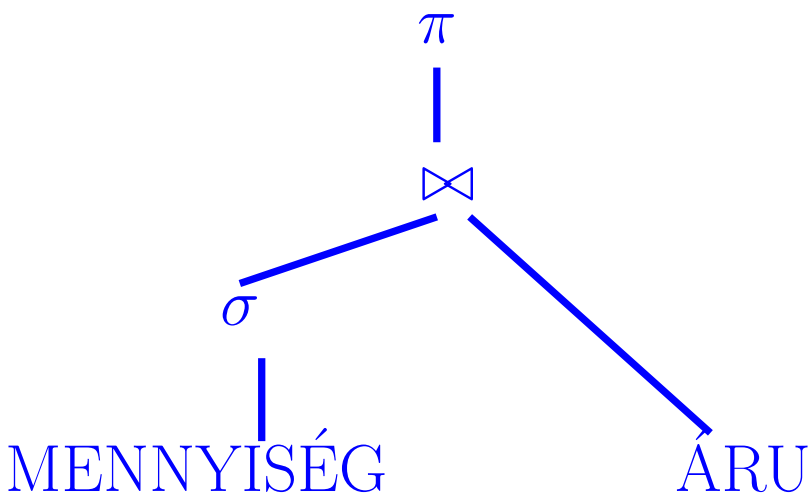
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right) \Longrightarrow$$

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG}\right) \bowtie \text{ÁRU}\right)$$





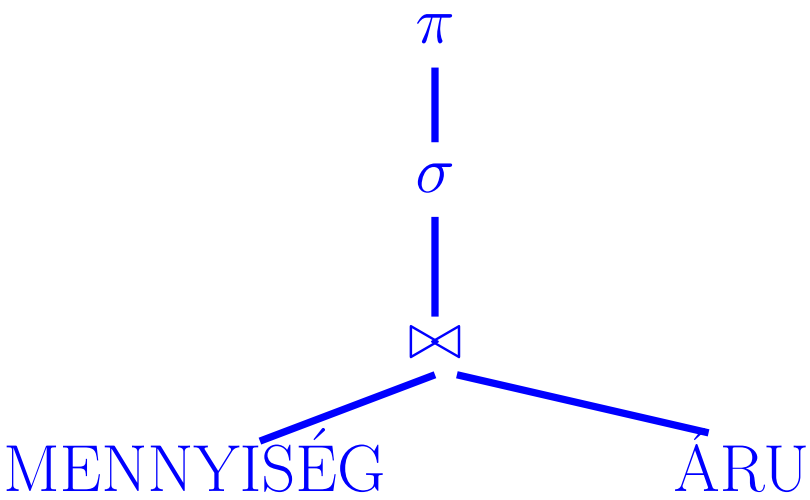
Kiválasztás tologatása

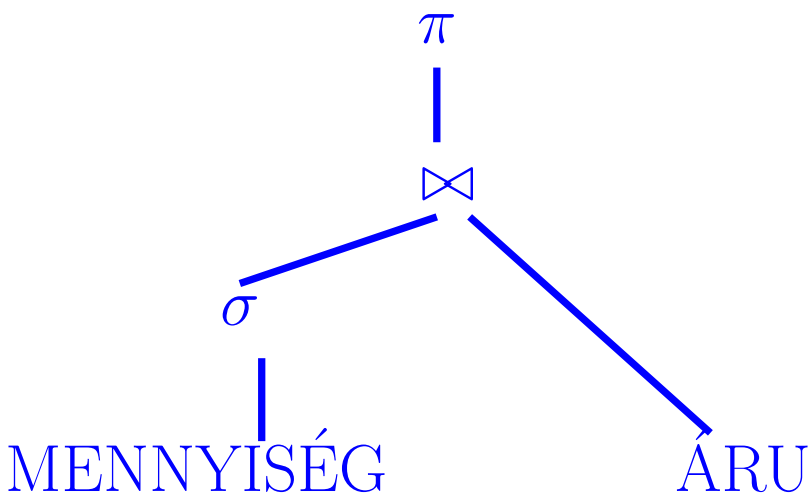
ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right) \Longrightarrow$$

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG}\right) \bowtie \text{ÁRU}\right)$$





Felhasznált azonosság:

$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$, ha minden C-beli attribútum szerepel R-ben.

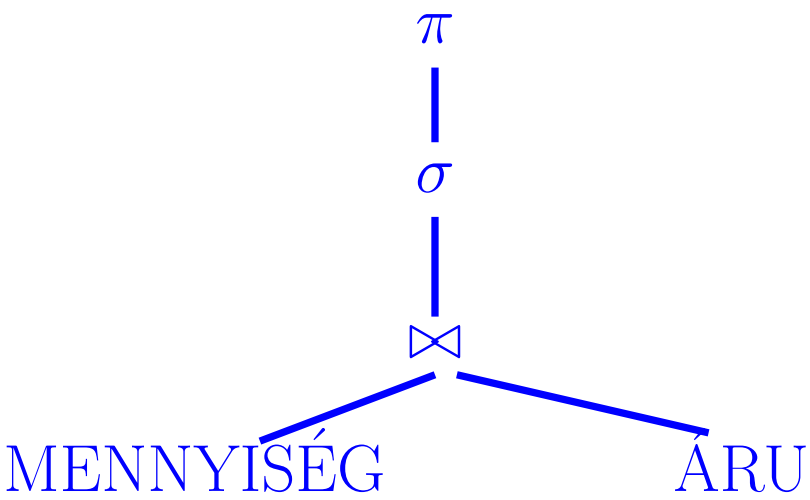
Kiválasztás tologatása

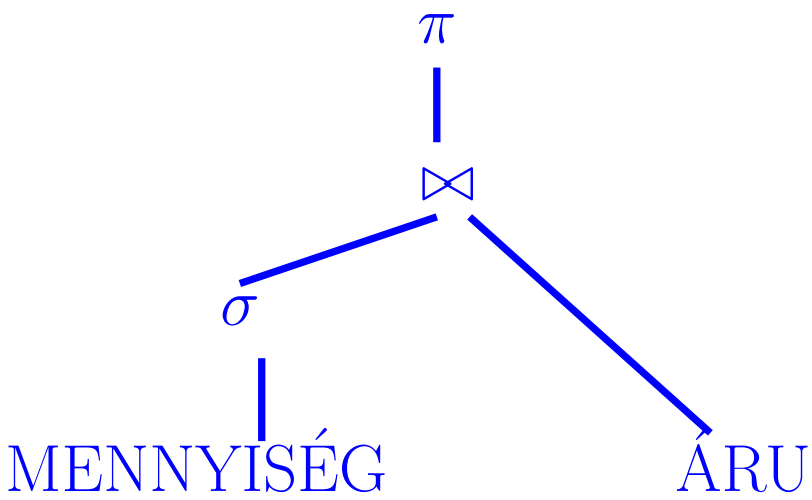
ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right) \Longrightarrow$$

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG}\right) \bowtie \text{ÁRU}\right)$$





Felhasznált azonosság:

$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$, ha minden C-beli attribútum szerepel R-ben.

Hasonló azonosságok:

$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$,

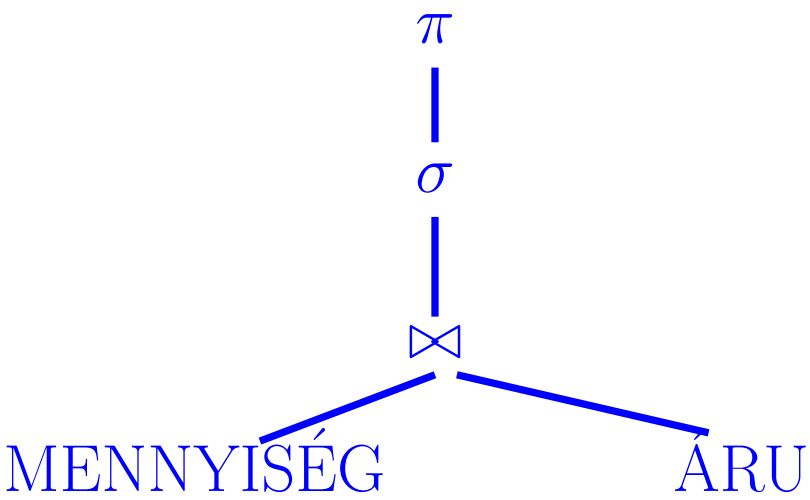
Kiválasztás tologatása

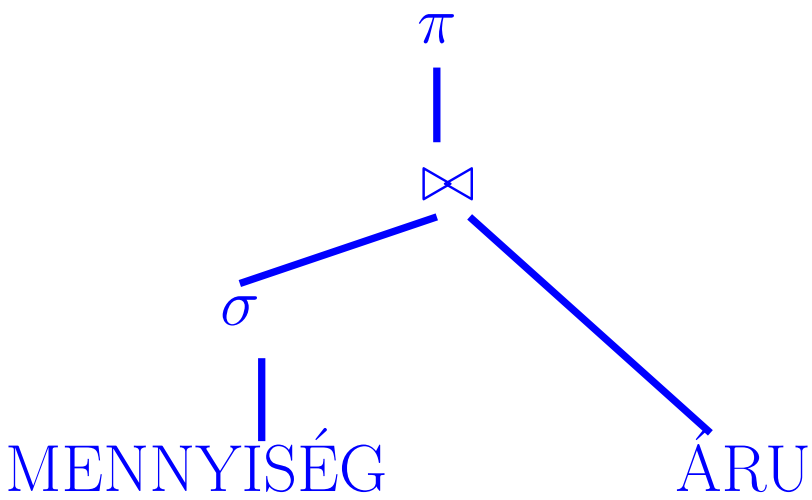
ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right) \Longrightarrow$$

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG}\right) \bowtie \text{ÁRU}\right)$$





Felhasznált azonosság:

$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$, ha minden C-beli attribútum szerepel R-ben.

Hasonló azonosságok:

$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$,

$\sigma_C(R \times S) = \sigma_C(R) \times S$, ha minden C-beli attribútum szerepel R-ben.

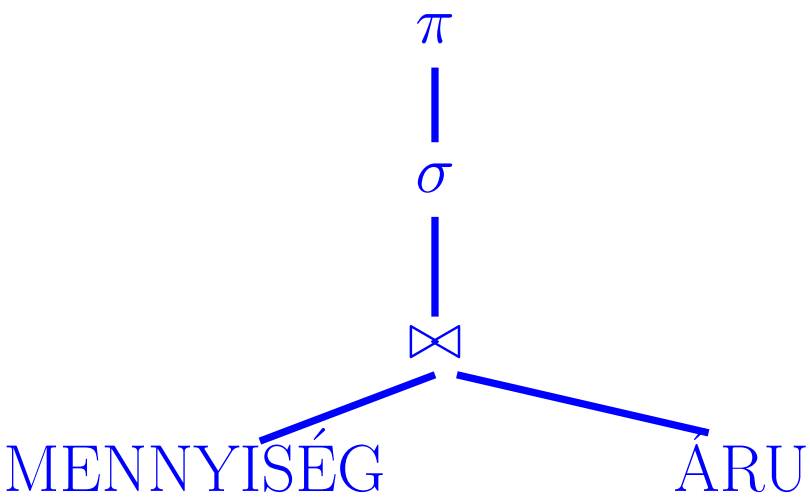
Kiválasztás tologatása

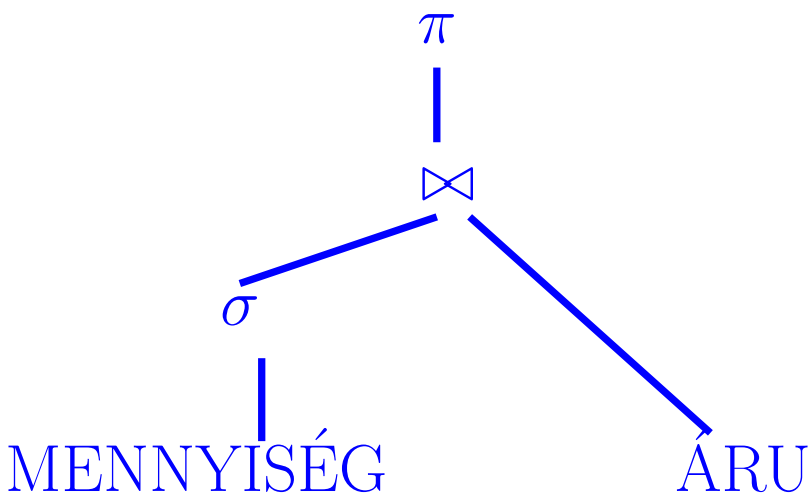
ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right) \Longrightarrow$$

$$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG}\right) \bowtie \text{ÁRU}\right)$$





Felhasznált azonosság:

$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$, ha minden C-beli attribútum szerepel R-ben.

Hasonló azonosságok:

$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$,

$\sigma_C(R \times S) = \sigma_C(R) \times S$, ha minden C-beli attribútum szerepel R-ben.

$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie \sigma_C(S)$, ha minden C-beli attribútum szerepel R-ben és S-ben is.

Kiválasztás tologatása

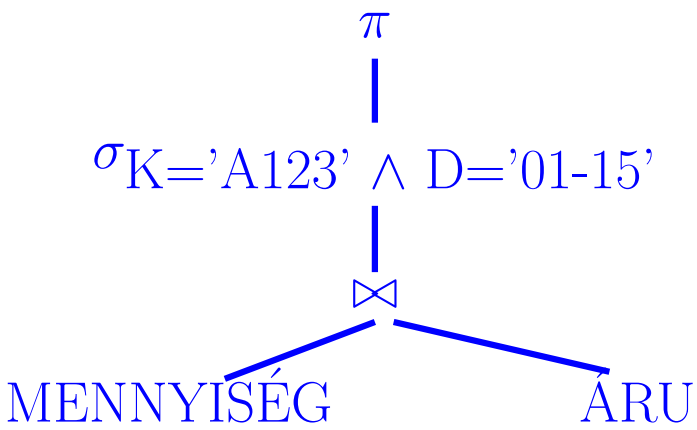
Összetett C szétszedhető:
$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$,

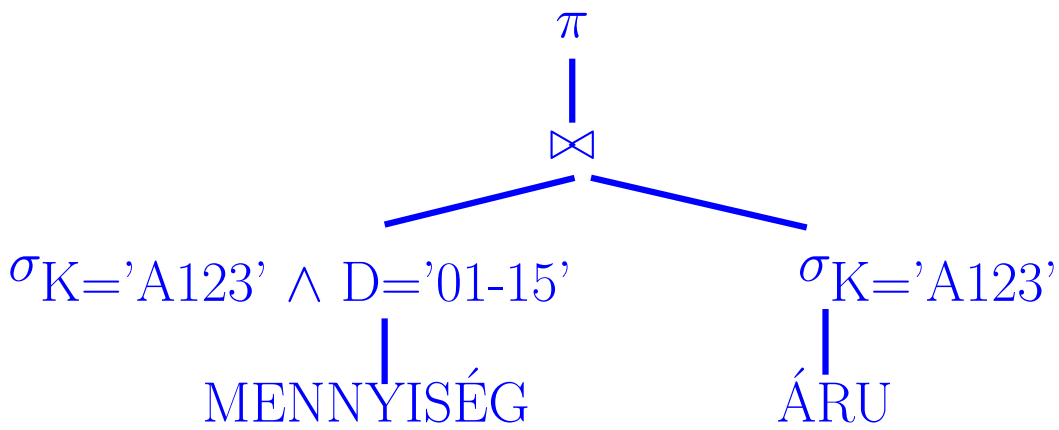
Kiválasztás tologatása

Összetett C szétszedhető:
$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$,
$\sigma_{C_1 \vee C_2}(R) = \sigma_{C_1}(R) \cup_H \sigma_{C_2}(R)$, ha R nem multihalmaz.

Kiválasztás tologatása

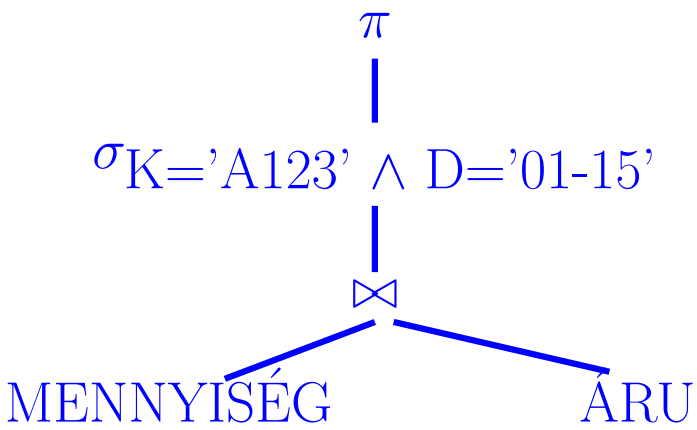
Összetett C szétszedhető:
$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$,
$\sigma_{C_1 \vee C_2}(R) = \sigma_{C_1}(R) \cup_H \sigma_{C_2}(R)$, ha R nem multihalmaz.

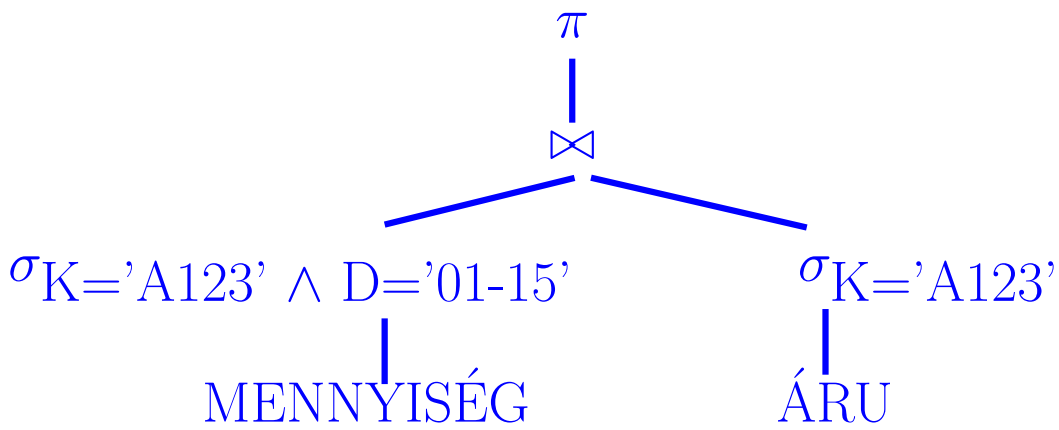




Kiválasztás tologatása

Összetett C szétszedhető:
$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$,
$\sigma_{C_1 \vee C_2}(R) = \sigma_{C_1}(R) \cup_H \sigma_{C_2}(R)$, ha R nem multihalmaz.





Lehet, hogy érdemes előbb feltolni, aztán le.

Más műveletekre vonatkozó szabályok

Hasonló szabályok projekcióra (π), duplikációk kiszűrésére (δ) és aggregációra (γ) is vannak, de ezek nem annyira csökkentik a műveletigényt. De csak olyan attribútumot lehet eltüntetni, amire nem hivatkozunk feljebb.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(R))$, ahol M az R olyan attribútumai, hogy vagy összekapcsolási attribútum, vagy L-beli, N pedig ...

Más műveletekre vonatkozó szabályok

Hasonló szabályok projekcióra (π), duplikációk kiszűrésére (δ) és aggregációra (γ) is vannak, de ezek nem annyira csökkentik a műveletigényt. De csak olyan attribútumot lehet eltüntetni, amire nem hivatkozunk feljebb.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(R))$, ahol M az R olyan attribútumai, hogy vagy összekapcsolási attribútum, vagy L-beli, N pedig ...
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$

Más műveletekre vonatkozó szabályok

Hasonló szabályok projekcióra (π), duplikációk kiszűrésére (δ) és aggregációra (γ) is vannak, de ezek nem annyira csökkentik a műveletigényt. De csak olyan attribútumot lehet eltüntetni, amire nem hivatkozunk feljebb.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(R))$, ahol M az R olyan attribútumai, hogy vagy összekapcsolási attribútum, vagy L-beli, N pedig ...
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$

Más műveletekre vonatkozó szabályok

Hasonló szabályok projekcióra (π), duplikációk kiszűrésére (δ) és aggregációra (γ) is vannak, de ezek nem annyira csökkentik a műveletigényt. De csak olyan attribútumot lehet eltüntetni, amire nem hivatkozunk feljebb.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(R))$, ahol M az R olyan attribútumai, hogy vagy összekapcsolási attribútum, vagy L-beli, N pedig ...
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
- $\delta(\gamma_L(R)) = \gamma_L(R)$

Más műveletekre vonatkozó szabályok

Hasonló szabályok projekcióra (π), duplikációk kiszűrésére (δ) és aggregációra (γ) is vannak, de ezek nem annyira csökkentik a műveletigényt. De csak olyan attribútumot lehet eltüntetni, amire nem hivatkozunk feljebb.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(R))$, ahol M az R olyan attribútumai, hogy vagy összekapcsolási attribútum, vagy L-beli, N pedig ...
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
- $\delta(\gamma_L(R)) = \gamma_L(R)$

De pl. δ nem tolható át $\cup_M$, π-n.

Más műveletekre vonatkozó szabályok

Hasonló szabályok projekcióra (π), duplikációk kiszűrésére (δ) és aggregációra (γ) is vannak, de ezek nem annyira csökkentik a műveletigényt. De csak olyan attribútumot lehet eltüntetni, amire nem hivatkozunk feljebb.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(R))$, ahol M az R olyan attribútumai, hogy vagy összekapcsolási attribútum, vagy L-beli, N pedig ...
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
- $\delta(\gamma_L(R)) = \gamma_L(R)$

De pl. δ nem tolható át $\cup_M, \pi$-n.
Még egy fontos kérdés az alkérdések kezelése, de erről most nem szólunk.

Összefoglalás

Sok ilyen szabály alkalmazásával többféle logikai terv előállítható.

Összefoglalás

Sok ilyen szabály alkalmazásával többféle logikai terv előállítható.
Ezeknek megbecsüljük a költségét és választunk egyet. Ehhez készítünk fizikai tervet.

Összefoglalás

Sok ilyen szabály alkalmazásával többféle logikai terv előállítható.
Ezeknek megbecsüljük a költségét és választunk egyet. Ehhez készítünk fizikai tervet.
Jobb rendszerekben ez automatikus. Ilyenekben kérdéses, hogy a lekérdezés beírásakor mire kell figyelni.

Összefoglalás

Sok ilyen szabály alkalmazásával többféle logikai terv előállítható.
Ezeknek megbecsüljük a költségét és választunk egyet. Ehhez készítünk fizikai tervet.
Jobb rendszerekben ez automatikus. Ilyenekben kérdéses, hogy a lekérdezés beírásakor mire kell figyelni.
Inkább olyan egyszerűsítéseket érdemes csak elvégezni, ami a függések következménye, mert ezeket nehezebben lehet automatizálni.

Összefoglalás

Sok ilyen szabály alkalmazásával többféle logikai terv előállítható.
Ezeknek megbecsüljük a költségét és választunk egyet. Ehhez készítünk fizikai tervet.
Jobb rendszerekben ez automatikus. Ilyenekben kérdéses, hogy a lekérdezés beírásakor mire kell figyelni.
Inkább olyan egyszerűsítéseket érdemes csak elvégezni, ami a függések következménye, mert ezeket nehezebben lehet automatizálni.
Pl. Oracle-ban van rá mód, hogy megnézzük mi a logikai és fizikai terv és meg lehet adni, hogy pontosan mit csináljon.

Fizikai szervezés, tárkezelés

Célja: a rekordokból (egy rekord = a reláció egy sora) álló állomány kezelése úgy, hogy az adatokhoz való hozzáférés gyors legyen.

Fizikai szervezés, tárkezelés

Célja: a rekordokból (egy rekord = a reláció egy sora) álló állomány kezelése úgy, hogy az adatokhoz való hozzáférés gyors legyen.

Fontos jellemzők:

- külső táras adatkezelés, mert sok az adat $\Longrightarrow$ ha valamivel dolgozni akarunk, akkor be kell hozni a belső memóriába $\Longrightarrow$ a költséget a beolvasás/kiírás jelenti

Fizikai szervezés, tárkezelés

Célja: a rekordokból (egy rekord = a reláció egy sora) álló állomány kezelése úgy, hogy az adatokhoz való hozzáférés gyors legyen.

Fontos jellemzők:

- külső táras adatkezelés, mert sok az adat $\Longrightarrow$ ha valamivel dolgozni akarunk, akkor be kell hozni a belső memóriába $\Longrightarrow$ a költséget a beolvasás/kiírás jelenti $\Longrightarrow$ az I/O műveletek számára akarunk optimalizálni

Fizikai szervezés, tárkezelés

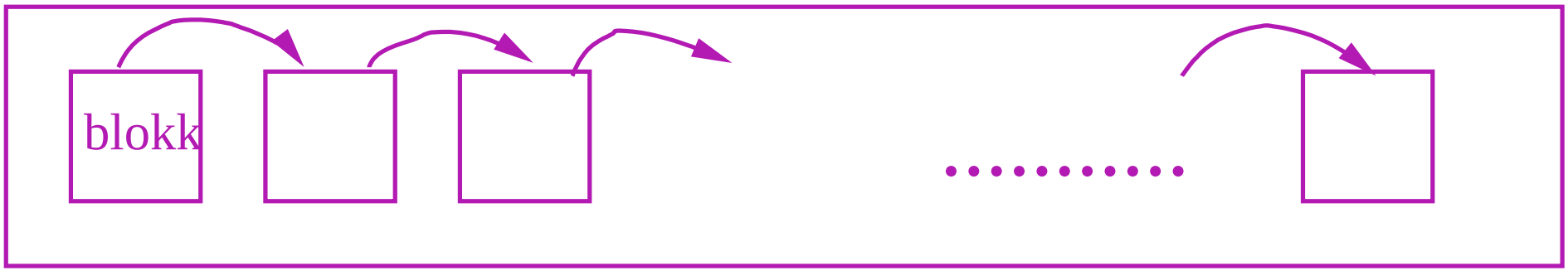
Célja: a rekordokból (egy rekord = a reláció egy sora) álló állomány kezelése úgy, hogy az adatokhoz való hozzáférés gyors legyen.

Fontos jellemzők:

- külső táras adatkezelés, mert sok az adat $\Longrightarrow$ ha valamivel dolgozni akarunk, akkor be kell hozni a belső memóriába $\Longrightarrow$ a költséget a beolvasás/kiírás jelenti $\Longrightarrow$ az I/O műveletek számára akarunk optimalizálni
- a műveletek, amiket gyorsan meg kell tudni csinálni: rekordok beillesztése, törlése, módosítása, keresése

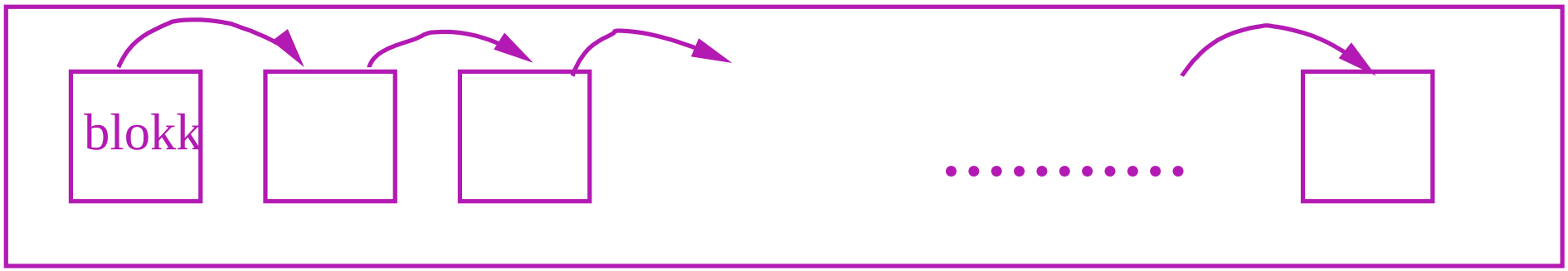
Az állomány felépítése

Az adatállomány a külső táron van, blokkok (lapok) elérésfolytonos sorozatán.



Az állomány felépítése

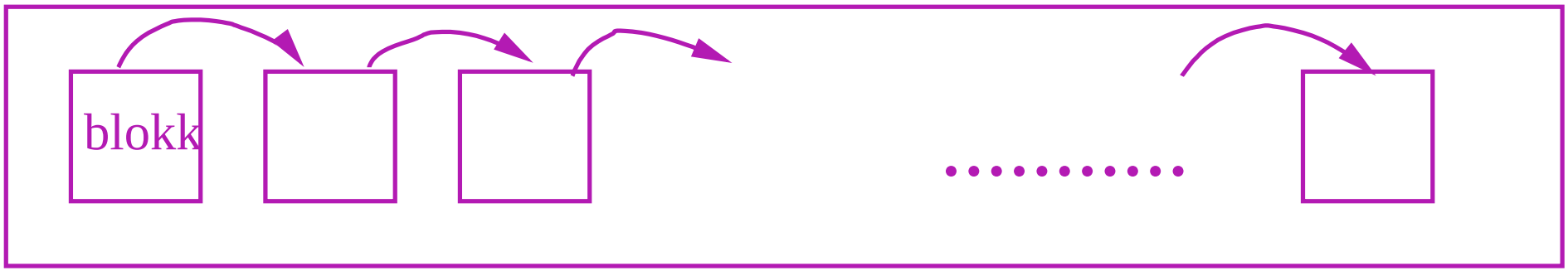
Az adatállomány a külső táron van, blokkok (lapok) elérésfolytonos sorozatán.



- egyszerre egy blokk írható ki/olvasható be

Az állomány felépítése

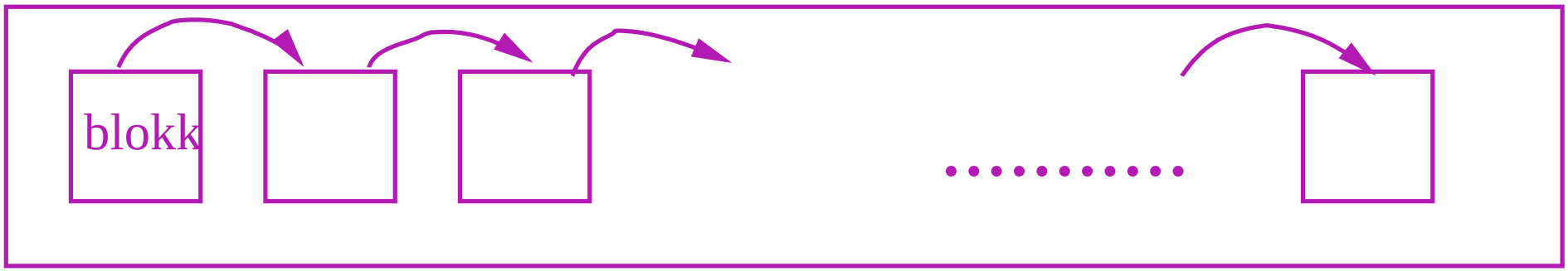
Az adatállomány a külső táron van, blokkok (lapok) elérésfolytonos sorozatán.



- egyszerre egy blokk írható ki/olvasható be
- blokk méret fix (ált. 2^{10}, 2^{12} byte)

Az állomány felépítése

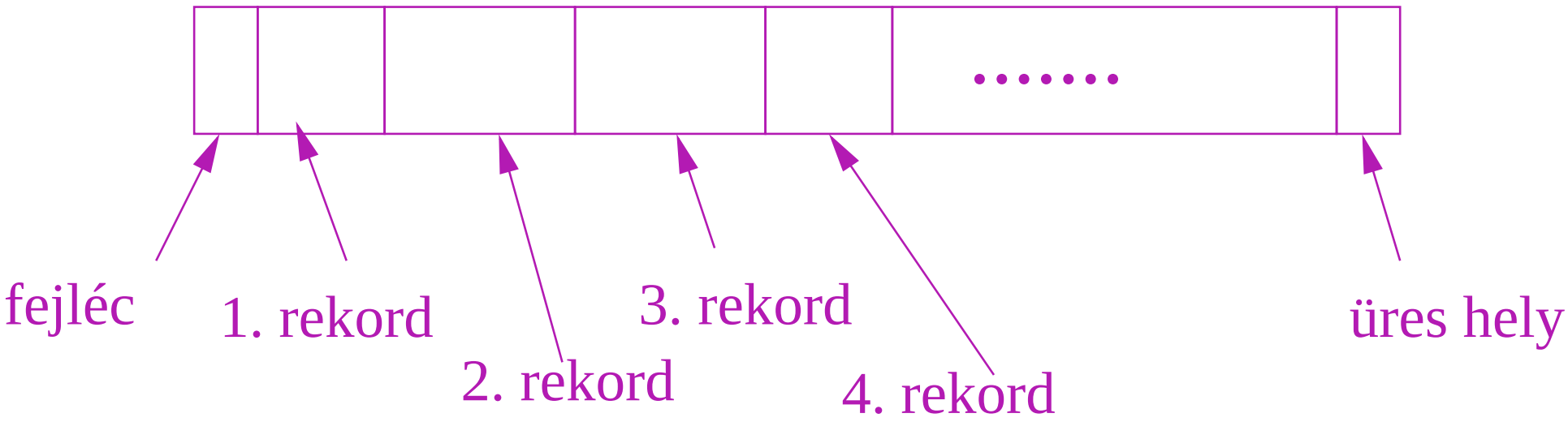
Az adatállomány a külső táron van, blokkok (lapok) elérésfolytonos sorozatán.



- egyszerre egy blokk írható ki/olvasható be
- blokk méret fix (ált. 2^{10}, 2^{12} byte)
- az operációs rendszer tartja nyilván, hogy melyik reláció rekordjai hol vannak és ő biztosítja az elérésfolytonosságot is

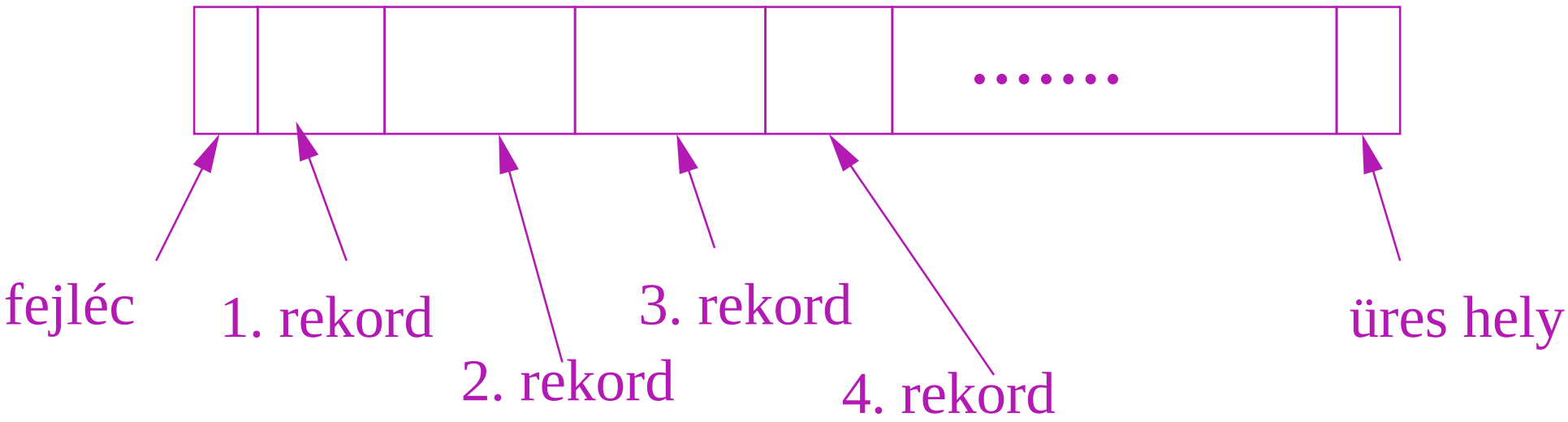
Blokkokról általában

Tipikus blokk



Blokkokról általában

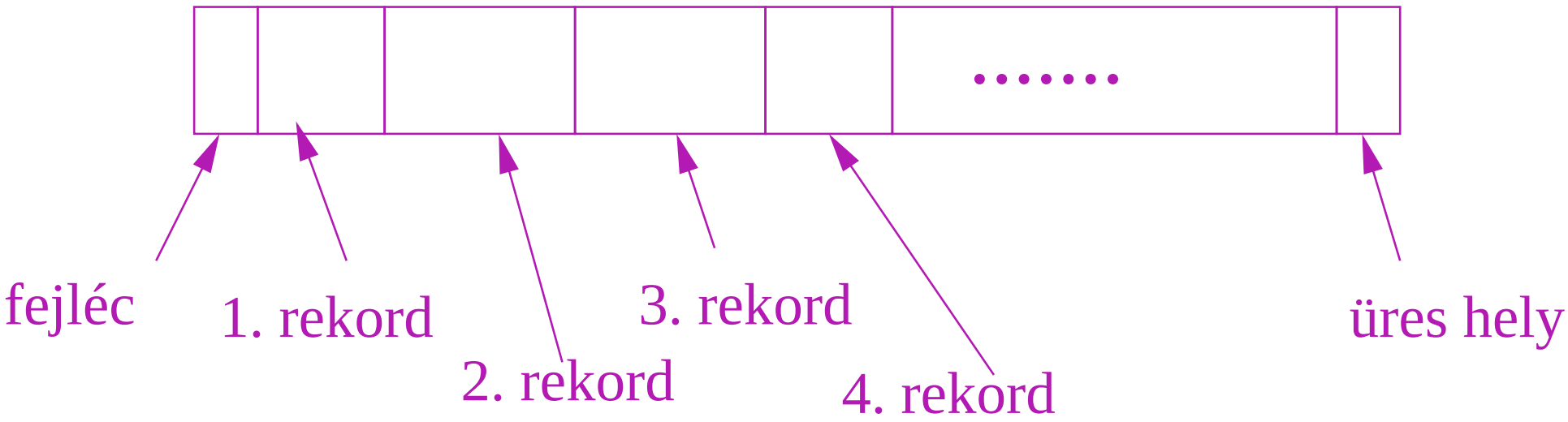
Tipikus blokk



A fejléc tartalmazza a blokkra vonatkozó infókat, (pl: melyik relációhoz tartozik, mennyi a szabad hely benne, hol kezdődik); ezután jönnek a rekordok egymás után, a végén általában marad üres hely.

Blokkokról általában

Tipikus blokk

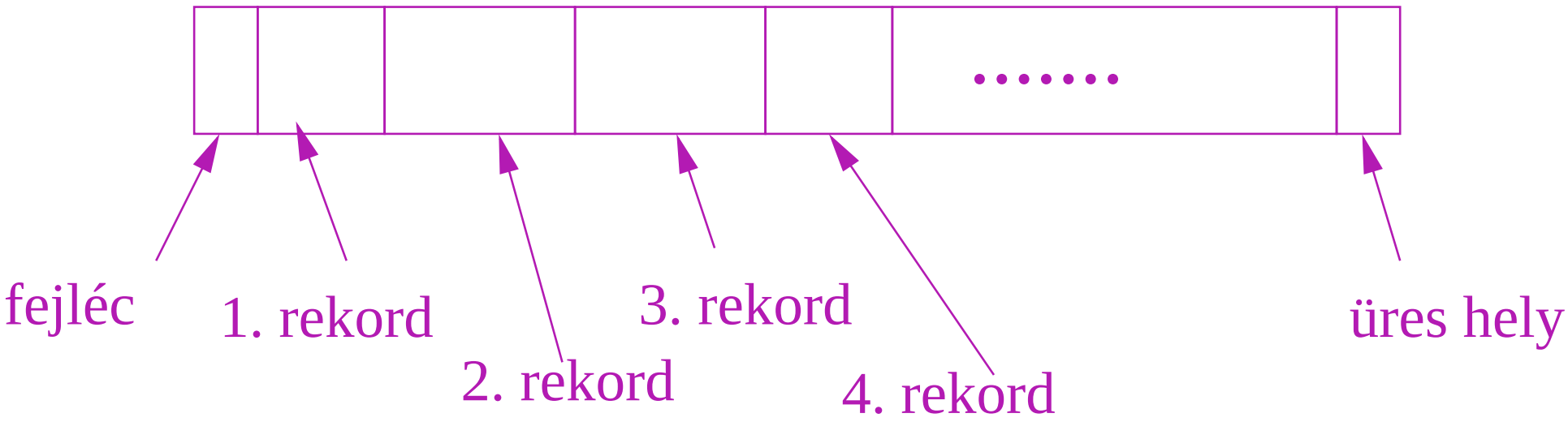


A fejléc tartalmazza a blokkra vonatkozó infókat, (pl: melyik relációhoz tartozik, mennyi a szabad hely benne, hol kezdődik); ezután jönnek a rekordok egymás után, a végén általában marad üres hely.

Fontos feltevés: rekordok blokkhatárt nem lépnek át, ezért általában van üres, fel nem használható hely a blokkok végén.

Blokkokról általában

Tipikus blokk

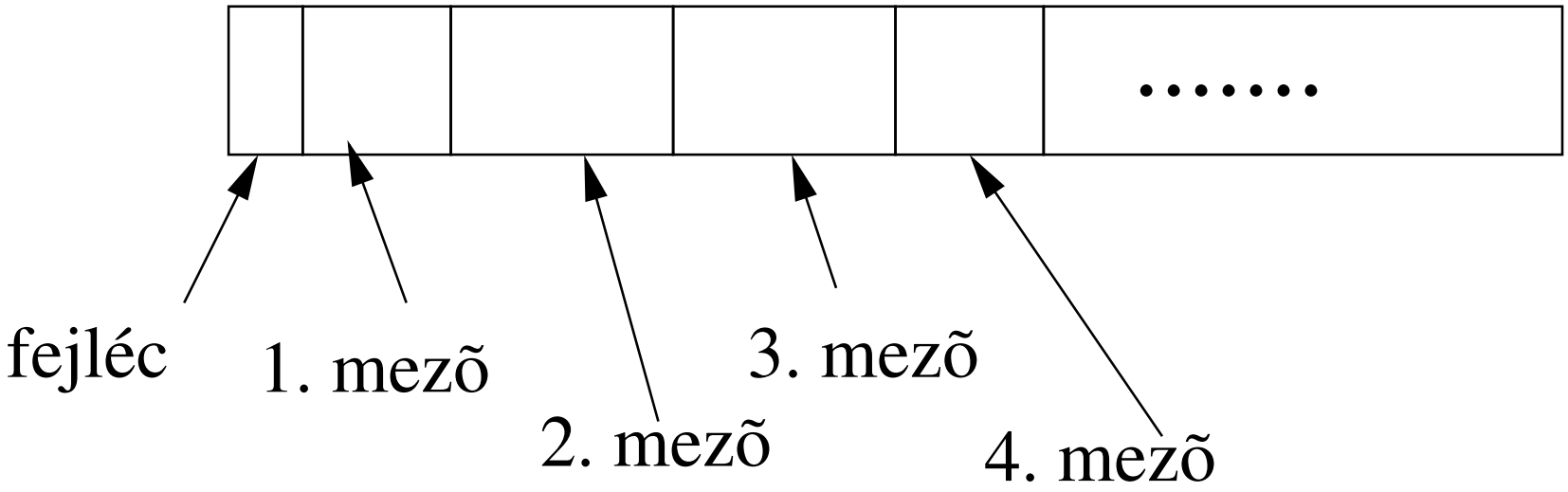


A fejléc tartalmazza a blokkra vonatkozó infókat, (pl: melyik relációhoz tartozik, mennyi a szabad hely benne, hol kezdődik); ezután jönnek a rekordok egymás után, a végén általában marad üres hely.

Fontos feltevés: rekordok blokkhatárt nem lépnek át, ezért általában van üres, fel nem használható hely a blokkok végén. (Ha nagyok a rekordok, pl. képfile-ok, és mégis át kell lépni laphatárt, akkor extra technikák kellenek, de ezzel most nem foglalkozunk.)

Kötött formátum

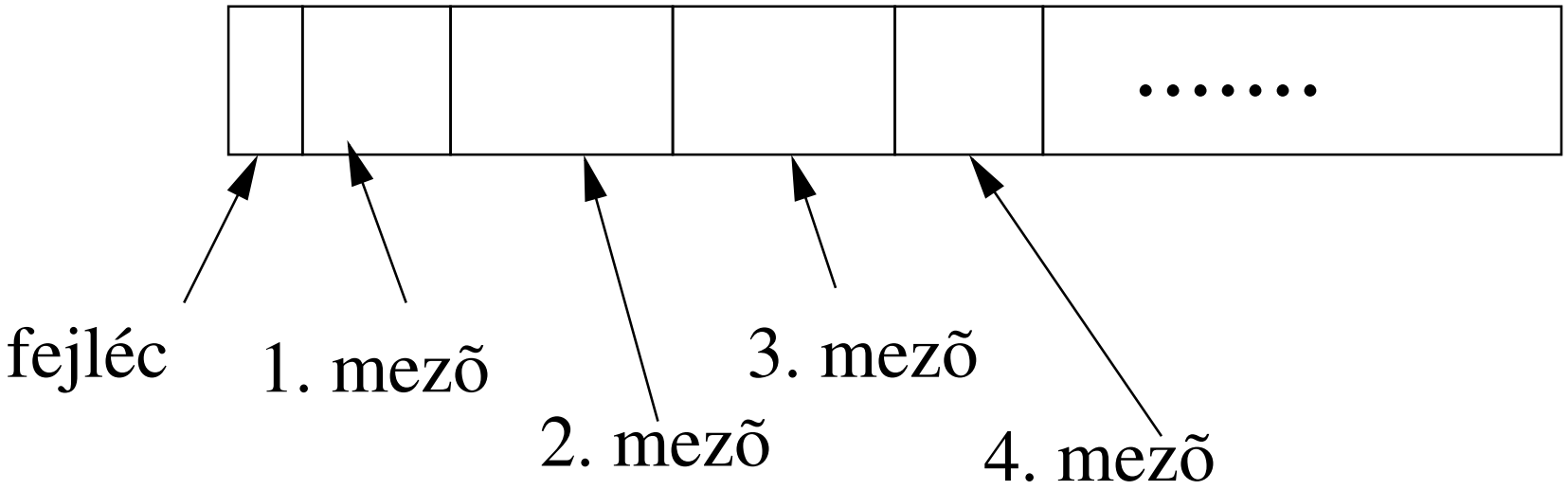
Ekkor a mezők száma, mérete, típusa és sorrendje fix



Rekordok típusai

Kötött formátum

Ekkor a mezők száma, mérete, típusa és sorrendje fix



Fejléc:

- a rekord kezelésével kapcsolatos infók: törölt-e, melyik relációhoz tartozik
- a mezők típusa
- időbélyeg (mikor módosult utoljára)

Rekordok típusai

Változó formátum

- Mezők hossza esetleg nem fix (szöveget tartalmazó adatbázisok)

Rekordok típusai

Változó formátum

- Mezők hossza esetleg nem fix (szöveget tartalmazó adatbázisok)
- Ismétlődő mezők lehetnek, és az ismétlések száma nem fix vagy pedig többértékű mezők (szereplők felsorolása filmnél)

Rekordok típusai

Változó formátum

- Mezők hossza esetleg nem fix (szöveget tartalmazó adatbázisok)
- Ismétlődő mezők lehetnek, és az ismétlések száma nem fix vagy pedig többértékű mezők (szereplők felsorolása filmnél)

Ilyenkor bonyolultabb a fejléc, kevésbé lehet gazdaságosan/előre tervezhetően tárolni a rekordokat, ezért érjük el inkább, hogy ne legyen ez az eset:

Rekordok típusai

Változó formátum

- Mezők hossza esetleg nem fix (szöveget tartalmazó adatbázisok)
- Ismétlődő mezők lehetnek, és az ismétlések száma nem fix vagy pedig többértékű mezők (szereplők felsorolása filmnél)

Ilyenkor bonyolultabb a fejléc, kevésbé lehet gazdaságosan/előre tervezhetően tárolni a rekordokat, ezért érjük el inkább, hogy ne legyen ez az eset:
Vezessük vissza ezt az esetet a kötöttre, pl. mutatók alkalmazásával a problémás helyeken

Rekordok típusai

Változó formátum

- Mezők hossza esetleg nem fix (szöveget tartalmazó adatbázisok)
- Ismétlődő mezők lehetnek, és az ismétlések száma nem fix vagy pedig többértékű mezők (szereplők felsorolása filmnél)

Ilyenkor bonyolultabb a fejléc, kevésbé lehet gazdaságosan/előre tervezhetően tárolni a rekordokat, ezért érjük el inkább, hogy ne legyen ez az eset:
Vezessük vissza ezt az esetet a kötöttre, pl. mutatók alkalmazásával a problémás helyeken $\Longrightarrow$ Mostantól feltesszük, hogy a rekordok kötött formátumúak és hogy az egész állományon belül ugyanaz a formátum van.

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés
2. **kötött blokk/rekord:** mutathat rá mutató, ezért nem mozgatható el szabadon. Ez típusszinten adott, azaz ha egy reláció rekordjaira/blokkjaira mutathat mutató, akkor még akkor is kötöttnek számít, ha éppen nem mutat egyre se semmi.

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés
2. **kötött blokk/rekord:** mutathat rá mutató, ezért nem mozgatható el szabadon. Ez típusszinten adott, azaz ha egy reláció rekordjaira/blokkjaira mutathat mutató, akkor még akkor is kötöttnek számít, ha éppen nem mutat egyre se semmi.
3. **szabad blokk/rekord:** nem mutathat rá mutató

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés
2. **kötött blokk/rekord:** mutathat rá mutató, ezért nem mozgatható el szabadon. Ez típusszinten adott, azaz ha egy reláció rekordjaira/blokkjaira mutathat mutató, akkor még akkor is kötöttnek számít, ha éppen nem mutat egyre se semmi.
3. **szabad blokk/rekord:** nem mutathat rá mutató
4. **Kulcs, keresési kulcs** (néha csak kulcsnak hívjuk):
   - a rekordok mezőinek egy kitüntetett halmaza (a reláció attribútumainak egy részhalmaza)

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés
2. **kötött blokk/rekord:** mutathat rá mutató, ezért nem mozgatható el szabadon. Ez típusszinten adott, azaz ha egy reláció rekordjaira/blokkjaira mutathat mutató, akkor még akkor is kötöttnek számít, ha éppen nem mutat egyre se semmi.
3. **szabad blokk/rekord:** nem mutathat rá mutató
4. **Kulcs, keresési kulcs** (néha csak kulcsnak hívjuk):
   - a rekordok mezőinek egy kitüntetett halmaza (a reláció attribútumainak egy részhalmaza)
   - ez alapján megy a keresés (ezeknek az értékét adjuk meg és azokat a rekordokat (sorokat a relációban) keressük, amiknél pont ezek az értékek szerepelnek)

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés
2. **kötött blokk/rekord:** mutathat rá mutató, ezért nem mozgatható el szabadon. Ez típusszinten adott, azaz ha egy reláció rekordjaira/blokkjaira mutathat mutató, akkor még akkor is kötöttnek számít, ha éppen nem mutat egyre se semmi.
3. **szabad blokk/rekord:** nem mutathat rá mutató
4. **Kulcs, keresési kulcs** (néha csak kulcsnak hívjuk):
   - a rekordok mezőinek egy kitüntetett halmaza (a reláció attribútumainak egy részhalmaza)
   - ez alapján megy a keresés (ezeknek az értékét adjuk meg és azokat a rekordokat (sorokat a relációban) keressük, amiknél pont ezek az értékek szerepelnek)
   - a keresési kulcs nem egyezik meg feltétlenül a reláció egyik kulcsával sem (pl. név a telefonkönyvnél)

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés
2. **kötött blokk/rekord:** mutathat rá mutató, ezért nem mozgatható el szabadon. Ez típusszinten adott, azaz ha egy reláció rekordjaira/blokkjaira mutathat mutató, akkor még akkor is kötöttnek számít, ha éppen nem mutat egyre se semmi.
3. **szabad blokk/rekord:** nem mutathat rá mutató
4. **Kulcs, keresési kulcs** (néha csak kulcsnak hívjuk):
   - a rekordok mezőinek egy kitüntetett halmaza (a reláció attribútumainak egy részhalmaza)
   - ez alapján megy a keresés (ezeknek az értékét adjuk meg és azokat a rekordokat (sorokat a relációban) keressük, amiknél pont ezek az értékek szerepelnek)
   - a keresési kulcs nem egyezik meg feltétlenül a reláció egyik kulcsával sem (pl. név a telefonkönyvnél)
   - de az azért elvárás, hogy ne legyen nagyon sok egy-egy értékre illeszkedő rekord

Alapvető állományszervezési technikák

Milyen struktúrát hozzunk létre az adatok tárolására?

Alapvető állományszervezési technikák

Milyen struktúrát hozzunk létre az adatok tárolására?

Lehetőségek:

1. Szekvenciális tárolás
2. Hash
3. Indexek

Szekvenciális tárolás

Nincs semmi struktúra, lineárisan töltjük fel az állományt, az új rekord az első alkalmas szabad helyre kerül.

Szekvenciális tárolás

Nincs semmi struktúra, lineárisan töltjük fel az állományt, az új rekord az első alkalmas szabad helyre kerül.

- **keresés**: egyesével beolvassuk a lapokat a belső memóriába, lineáris keresés, ha N lap van, akkor átlagosan $\frac{N}{2}$ I/O művelet

Szekvenciális tárolás

Nincs semmi struktúra, lineárisan töltjük fel az állományt, az új rekord az első alkalmas szabad helyre kerül.

- **keresés**: egyesével beolvassuk a lapokat a belső memóriába, lineáris keresés, ha N lap van, akkor átlagosan $\frac{N}{2}$ I/O művelet
- **törlés**: keresés, aztán törlésbittel (ha sok törlés volt, esetleg garbage collection időnként)

Szekvenciális tárolás

Nincs semmi struktúra, lineárisan töltjük fel az állományt, az új rekord az első alkalmas szabad helyre kerül.

- **keresés**: egyesével beolvassuk a lapokat a belső memóriába, lineáris keresés, ha N lap van, akkor átlagosan $\frac{N}{2}$ I/O művelet
- **törlés**: keresés, aztán törlésbittel (ha sok törlés volt, esetleg garbage collection időnként)
- **beszúrás**: keresünk szabad helyet és oda rakjuk. Ehhez egyesével beolvassuk a blokkokat, ha van üres hely oda rakjuk, ha nincs sehol, akkor az állomány végére. Ha az utolsó lapra se fér: új lapot kérünk

Szekvenciális tárolás

Nincs semmi struktúra, lineárisan töltjük fel az állományt, az új rekord az első alkalmas szabad helyre kerül.

- **keresés**: egyesével beolvassuk a lapokat a belső memóriába, lineáris keresés, ha N lap van, akkor átlagosan $\frac{N}{2}$ I/O művelet
- **törlés**: keresés, aztán törlésbittel (ha sok törlés volt, esetleg garbage collection időnként)
- **beszúrás**: keresünk szabad helyet és oda rakjuk. Ehhez egyesével beolvassuk a blokkokat, ha van üres hely oda rakjuk, ha nincs sehol, akkor az állomány végére. Ha az utolsó lapra se fér: új lapot kérünk
- **módosítás**: keresés, majd ha befér az eredeti helyére, akkor oda teszem vissza a módosítás után, különben meg törlés és beszúrás

Szekvenciális tárolás

Nincs semmi struktúra, lineárisan töltjük fel az állományt, az új rekord az első alkalmas szabad helyre kerül.

- **keresés**: egyesével beolvassuk a lapokat a belső memóriába, lineáris keresés, ha N lap van, akkor átlagosan $\frac{N}{2}$ I/O művelet
- **törlés**: keresés, aztán törlésbittel (ha sok törlés volt, esetleg garbage collection időnként)
- **beszúrás**: keresünk szabad helyet és oda rakjuk. Ehhez egyesével beolvassuk a blokkokat, ha van üres hely oda rakjuk, ha nincs sehol, akkor az állomány végére. Ha az utolsó lapra se fér: új lapot kérünk
- **módosítás**: keresés, majd ha befér az eredeti helyére, akkor oda teszem vissza a módosítás után, különben meg törlés és beszúrás

Akkor jó így tárolni, ha kevés az adat. Előnye, hogy nem kell adatszerkezettel vesződni.

Adatbázisok elmélete

Hash (tördelés)

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

19. előadás

Hash (tördelés)

Külső táras tárolás miatt vödrös hasht kell használni. (Nyílt címzés nem menne.)

Hash (tördelés)

Külső táras tárolás miatt vödrös hasht kell használni. (Nyílt címzés nem menne.)

Alapvető szerkezet: B vödör, ezekbe rakjuk majd a rekordokat. A keresési kulcs (K) értékétől függően.

Hash (tördelés)

Külső táras tárolás miatt vödrös hasht kell használni. (Nyílt címzés nem menne.)

Alapvető szerkezet: B vödör, ezekbe rakjuk majd a rekordokat. A keresési kulcs (K) értékétől függően.

Adott egy hash függvény, ami leképezi a tárolandó rekordokat (a keresési kulcsokat) a $[0, B-1]$ intervallum egész értékeire, azaz $h: \ K \rightarrow h(K) \in [0, B-1]$

Hash (tördelés)

Külső táras tárolás miatt vödrös hasht kell használni. (Nyílt címzés nem menne.)

Alapvető szerkezet: B vödör, ezekbe rakjuk majd a rekordokat. A keresési kulcs (K) értékétől függően.

Adott egy hash függvény, ami leképezi a tárolandó rekordokat (a keresési kulcsokat) a $[0, B-1]$ intervallum egész értékeire, azaz $h: \ K \rightarrow h(K) \in [0, B-1]$
Ez adja meg, hogy egy rekord melyik vödörbe kerüljön.

Hash (tördelés)

Külső táras tárolás miatt vödrös hasht kell használni. (Nyílt címzés nem menne.)

Alapvető szerkezet: B vödör, ezekbe rakjuk majd a rekordokat. A keresési kulcs (K) értékétől függően.

Adott egy hash függvény, ami leképezi a tárolandó rekordokat (a keresési kulcsokat) a $[0, B-1]$ intervallum egész értékeire, azaz $h: \ K \to h(K) \in [0, B-1]$
Ez adja meg, hogy egy rekord melyik vödörbe kerüljön.
(A lehetséges K-k, a keresési kulcsok értékei, egy nagy univerzumból kerülnek ki, de az összes előfordulásuk száma ennél jóval kisebb. B-t úgy választjuk meg, hogy a várható blokkszámmal legyen nagyjából egyenlő)

Hash (tördelés)

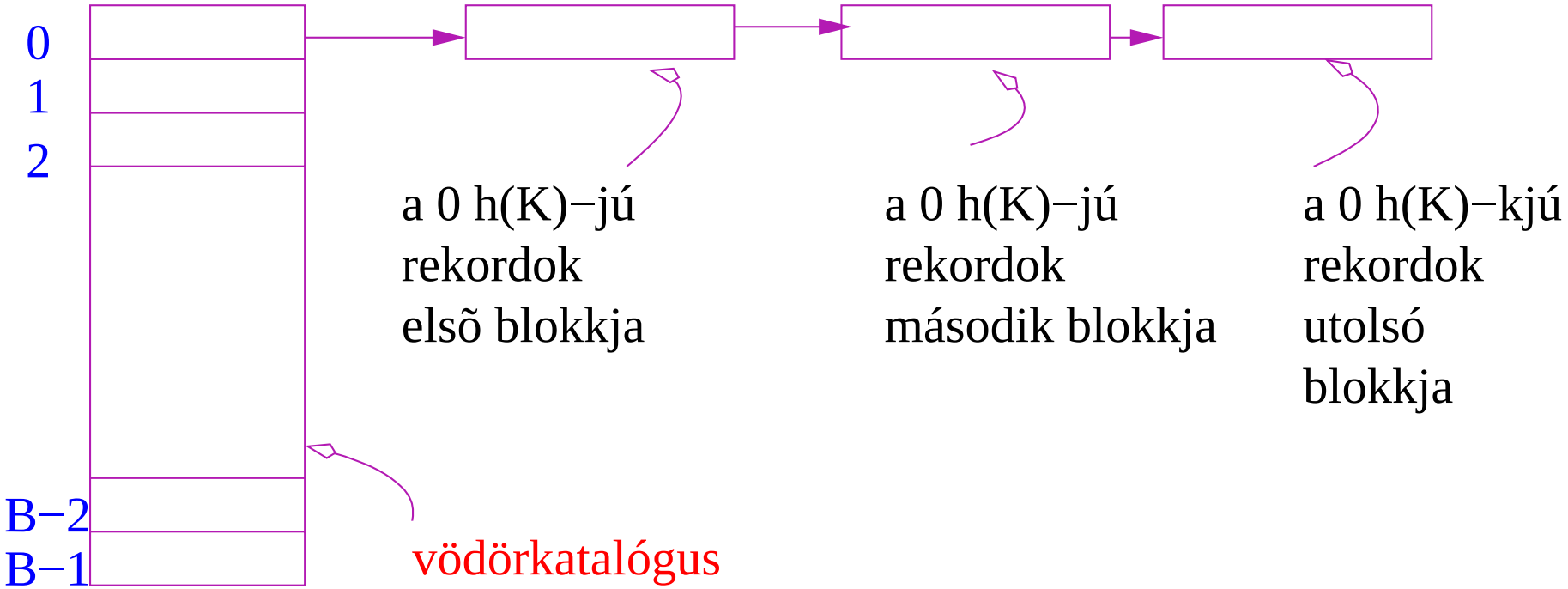
Külső táras tárolás miatt vödrös hasht kell használni. (Nyílt címzés nem menne.)

Alapvető szerkezet: B vödör, ezekbe rakjuk majd a rekordokat. A keresési kulcs (K) értékétől függően.

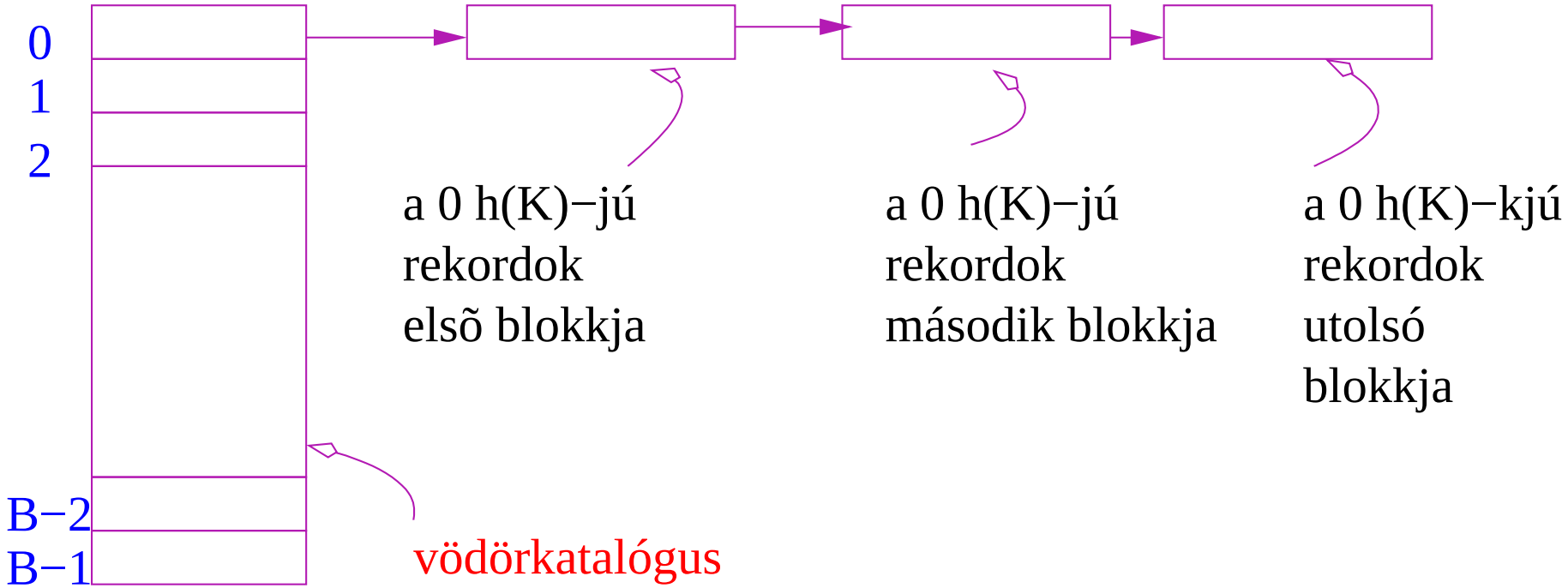
Adott egy hash függvény, ami leképezi a tárolandó rekordokat (a keresési kulcsokat) a $[0, B-1]$ intervallum egész értékeire, azaz $h: \ K \to h(K) \in [0, B-1]$
Ez adja meg, hogy egy rekord melyik vödörbe kerüljön.
(A lehetséges K-k, a keresési kulcsok értékei, egy nagy univerzumból kerülnek ki, de az összes előfordulásuk száma ennél jóval kisebb. B-t úgy választjuk meg, hogy a várható blokkszámmal legyen nagyjából egyenlő)

Elvárások a hash-függvénnyel szemben: gyorsan számolható legyen, kevés ütközést okozzon. Jó pl. a szorzómódszer ($h(K) = K \pmod M$) vagy az osztómódszer.

Hash

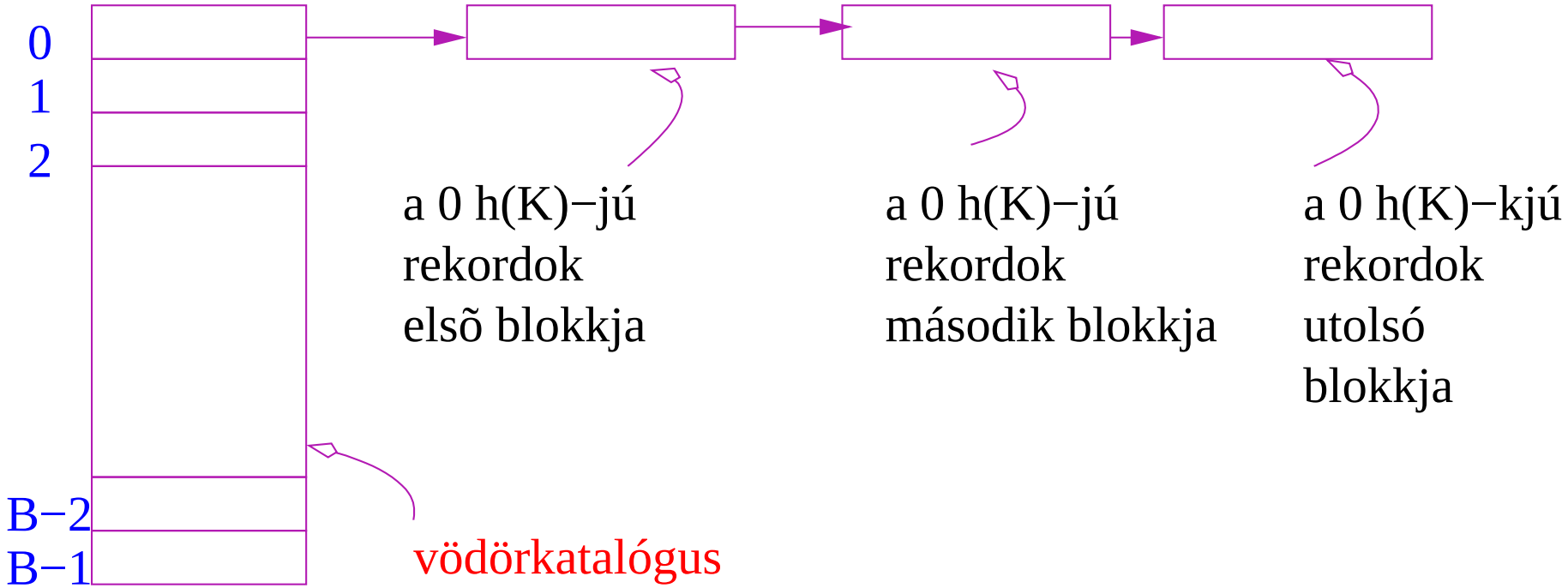

0
1
2
B−2
B−1
a 0 h(K)−jú rekordok elsõ blokkja
a 0 h(K)−jú rekordok második blokkja
a 0 h(K)−kjú rekordok utolsó blokkja
vödörkatalógus

Hash



Vödörkatalógus: tipikusan a belső memóriában tároljuk, ebben csak B darab mutató van, ez alapján tudjuk, hogy adott $h(K)$ esetén hol van az első blokkja a h(K) hashértékű rekordoknak.

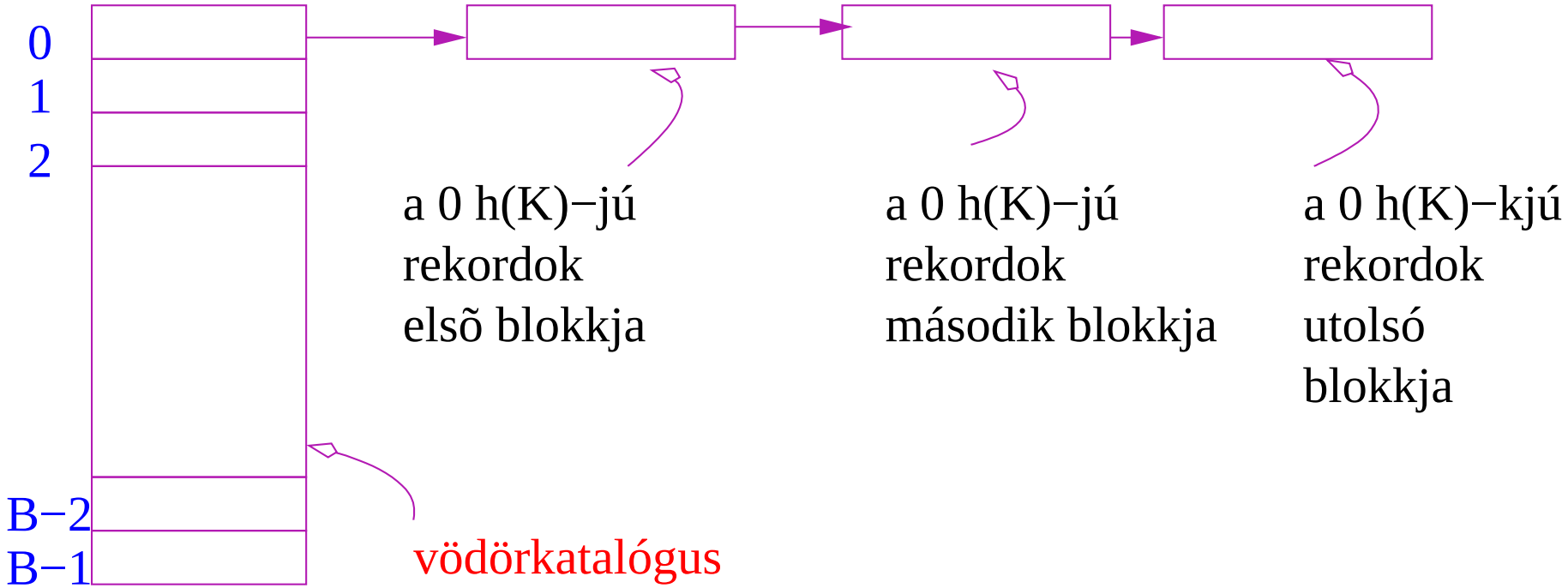
Hash



Vödörkatalógus: tipikusan a belső memóriában tároljuk, ebben csak B darab mutató van, ez alapján tudjuk, hogy adott $h(K)$ esetén hol van az első blokkja a h(K) hashértékű rekordoknak.

Egy vödör: azonos h(K)-jú rekordok halmaza, ezek néhány (jó esetben egy, de esetleg sok) blokkban vannak. Az egy vödörbeli blokkok között mutatókon tudunk mozogni.

Hash



Vödörkatalógus: tipikusan a belső memóriában tároljuk, ebben csak B darab mutató van, ez alapján tudjuk, hogy adott $h(K)$ esetén hol van az első blokkja a h(K) hashértékű rekordoknak.

Egy vödör: azonos h(K)-jú rekordok halmaza, ezek néhány (jó esetben egy, de esetleg sok) blokkban vannak. Az egy vödörbeli blokkok között mutatókon tudunk mozogni. A vödrön belül rendezettség semmi nincs, a rekordok az érkezési sorrendjükben vannak.

Műveletek

- **keresés**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödröt végigkeressük szekvenciálisan

Műveletek

- **keresés**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödröt végigkeressük szekvenciálisan
- **beszúrás**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödörbe rakjuk be a rekordot az első szabad helyre

Műveletek

- **keresés**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödröt végigkeressük szekvenciálisan
- **beszúrás**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödörbe rakjuk be a rekordot az első szabad helyre
- **törlés**: keresés, majd törlésbittel

Műveletek

- **keresés**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödröt végigkeressük szekvenciálisan
- **beszúrás**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödörbe rakjuk be a rekordot az első szabad helyre
- **törlés**: keresés, majd törlésbittel

Költség: ha jól van megválasztva B (nem nő túlságosan az állomány), akkor átlagosan konstans I/O művelettel megvan minden (legrosszabb esetben azonban nagyon rossz is lehet, annyira, mint a szekvenciális szervezés). Akkor jó, ha rövid blokkláncokból állnak a vödrök, ezért kellett B-t annyinak választani, mint a várható blokkszám.

Műveletek

- **keresés**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödröt végigkeressük szekvenciálisan
- **beszúrás**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödörbe rakjuk be a rekordot az első szabad helyre
- **törlés**: keresés, majd törlésbittel

Költség: ha jól van megválasztva B (nem nő túlságosan az állomány), akkor átlagosan konstans I/O művelettel megvan minden (legrosszabb esetben azonban nagyon rossz is lehet, annyira, mint a szekvenciális szervezés). Akkor jó, ha rövid blokkláncokból állnak a vödrök, ezért kellett B-t annyinak választani, mint a várható blokkszám.

Baj: ha elrontjuk B választását, túl dinamikusan nő az állomány $\Longrightarrow$ hosszú blokkláncok, lassú műveletek

Növelhető hash

Kiküszöböli a vödrös hash azon hibáját, hogy nem tud alkalmazkodni a gyorsan növő állományhoz.

Növelhető hash

Kiküszöböli a vödrös hash azon hibáját, hogy nem tud alkalmazkodni a gyorsan növő állományhoz.

Fő elvek:

- kiszámoljuk $h(K)$-t, de az eredménynek csak az első (vagy utolsó) pár jegye számít abban, hogy melyik rekord hova kerül

Növelhető hash

Kiküszöböli a vödrös hash azon hibáját, hogy nem tud alkalmazkodni a gyorsan növő állományhoz.

Fő elvek:

- kiszámoljuk $h(K)$-t, de az eredménynek csak az első (vagy utolsó) pár jegye számít abban, hogy melyik rekord hova kerül
- dinamikusan változik, hogy hány bit számít és ezzel együtt az is, hogy hány vödör lesz

Növelhető hash

Kiküszöböli a vödrös hash azon hibáját, hogy nem tud alkalmazkodni a gyorsan növő állományhoz.

Fő elvek:

- kiszámoljuk $h(K)$-t, de az eredménynek csak az első (vagy utolsó) pár jegye számít abban, hogy melyik rekord hova kerül
- dinamikusan változik, hogy hány bit számít és ezzel együtt az is, hogy hány vödör lesz
- a vödrök mérete fix, tipikusan egy blokkból állnak. Így majd a műveletek 1 lapeléréssel menni fognak, egy kis belső memóriában való keresgélés után.

Növelhető hash

Jellemzők:

- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke

Növelhető hash

Jellemzők:

- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke
- 2^d bejegyzés lesz a vödörkatalógusban, mert 2^d darab d hosszú bitsorozat van

Növelhető hash

Jellemzők:

- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke
- 2^d bejegyzés lesz a vödörkatalógusban, mert 2^d darab d hosszú bitsorozat van
- a vödörkatalógus most is egy mutatókból álló (most éppen 2^d elemű) tömb

Növelhető hash

Jellemzők:

- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke
- 2^d bejegyzés lesz a vödörkatalógusban, mert 2^d darab d hosszú bitsorozat van
- a vödörkatalógus most is egy mutatókból álló (most éppen 2^d elemű) tömb
- a vödrök száma 2^d-nél lehet kevesebb is, ha több mutató is mutat ugyanarra a vödörre

Növelhető hash

Jellemzők:

- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke
- 2^d bejegyzés lesz a vödörkatalógusban, mert 2^d darab d hosszú bitsorozat van
- a vödörkatalógus most is egy mutatókból álló (most éppen 2^d elemű) tömb
- a vödrök száma 2^d-nél lehet kevesebb is, ha több mutató is mutat ugyanarra a vödörre

A műveletek közös vonása: kiszámoljuk $h(K)$-t d bitig és aztán a vödörkatalógus $h(K)$-hoz tartozó mutatója alapján elmegyünk abba a vödörbe (blokkhoz), ahol az ezen d bittel kezdődő $h(K)$ értékű rekordok vannak.

Növelhető hash

Jellemzők:

- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke
- 2^d bejegyzés lesz a vödörkatalógusban, mert 2^d darab d hosszú bitsorozat van
- a vödörkatalógus most is egy mutatókból álló (most éppen 2^d elemű) tömb
- a vödrök száma 2^d-nél lehet kevesebb is, ha több mutató is mutat ugyanarra a vödörre

A műveletek közös vonása: kiszámoljuk $h(K)$-t d bitig és aztán a vödörkatalógus $h(K)$-hoz tartozó mutatója alapján elmegyünk abba a vödörbe (blokkhoz), ahol az ezen d bittel kezdődő $h(K)$ értékű rekordok vannak.
Ha két mutató ugyanoda mutat, akkor különböző d hosszú bitsorozattal kezdődő $h(K)$ értékű rekordok ugyanoda kerülnek.

Növelhető hash

Jellemzők:

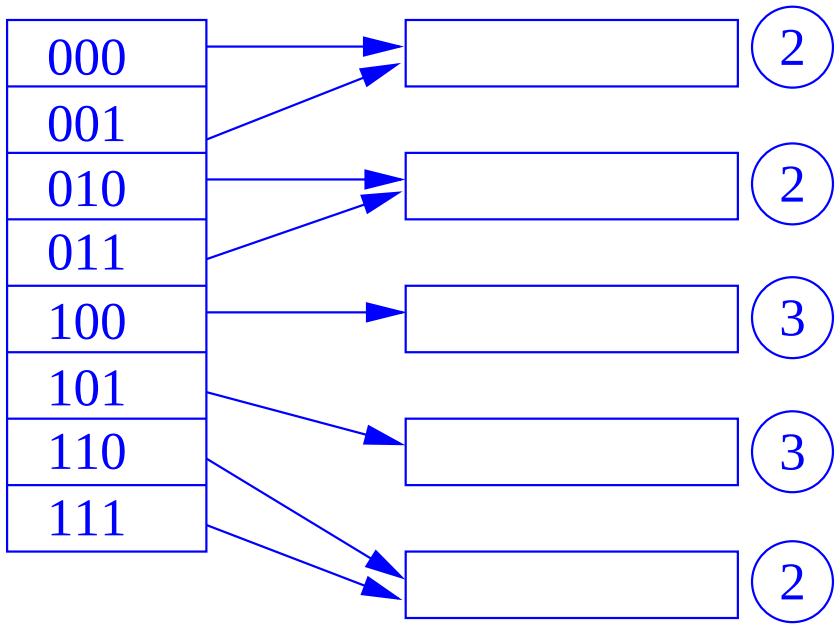
- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke
- 2^d bejegyzés lesz a vödörkatalógusban, mert 2^d darab d hosszú bitsorozat van
- a vödörkatalógus most is egy mutatókból álló (most éppen 2^d elemű) tömb
- a vödrök száma 2^d-nél lehet kevesebb is, ha több mutató is mutat ugyanarra a vödörre

A műveletek közös vonása: kiszámoljuk $h(K)$-t d bitig és aztán a vödörkatalógus $h(K)$-hoz tartozó mutatója alapján elmegyünk abba a vödörbe (blokkhoz), ahol az ezen d bittel kezdődő $h(K)$ értékű rekordok vannak.
Ha két mutató ugyanoda mutat, akkor különböző d hosszú bitsorozattal kezdődő $h(K)$ értékű rekordok ugyanoda kerülnek.

Minden vödörnek van egy lokális mélysége, ezt a d' változó tárolja. Ez azt mutatja, hogy az ebbe a vödörbe kerülésnél az első hány darab bit számít. Természetesen $d' \leq d$ áll minden vödörre, $d' < d$ akkor van, ha több mutató ugyanoda mutat.

A struktúra felépítése

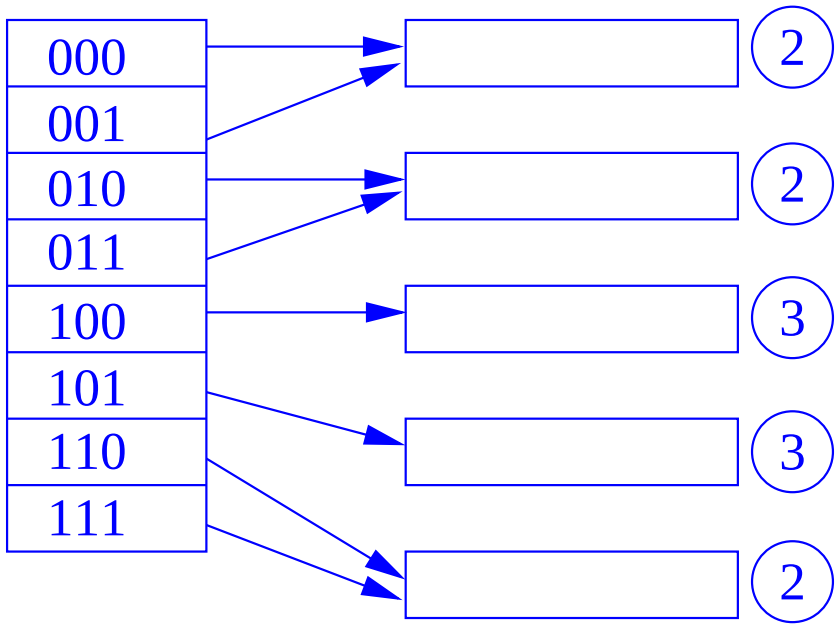
A struktúra vázlata $d = 3$ esetén:



Mivel $d = 3$, ezért az első három bitig számoljuk ki $h(K)$-t.

A struktúra felépítése

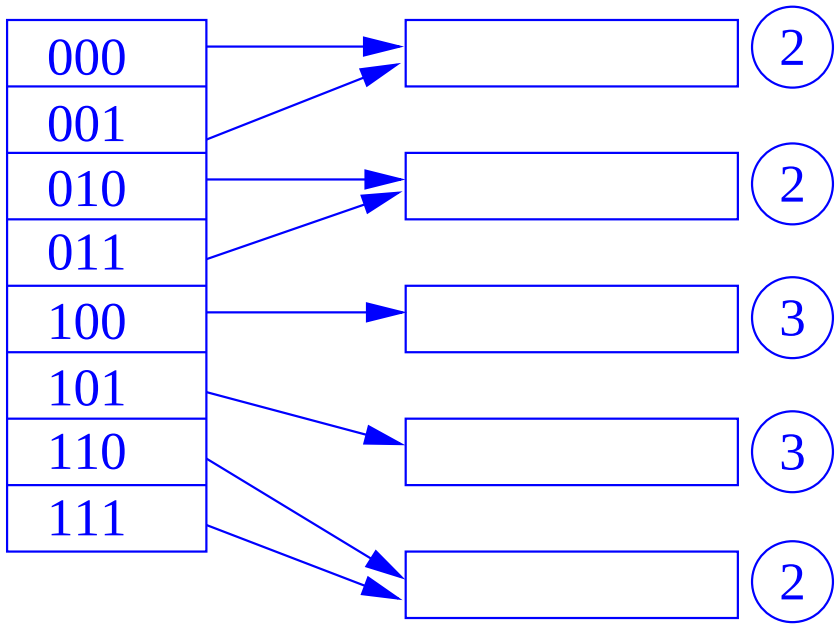
A struktúra vázlata $d = 3$ esetén:



Mivel $d = 3$, ezért az első három bitig számoljuk ki $h(K)$-t. Most összesen öt vödör van a lehetséges maximális nyolc helyett, mert azok a rekordok, amiknek a $h(K)$-ja 00-val kezdődik elférnek egy vödörben, itt már az első két bit alapján tudjuk, hogy melyik vödörbe kerül a rekord.

A struktúra felépítése

A struktúra vázlata $d = 3$ esetén:



Mivel $d = 3$, ezért az első három bitig számoljuk ki $h(K)$-t. Most összesen öt vödör van a lehetséges maximális nyolc helyett, mert azok a rekordok, amiknek a $h(K)$-ja 00-val kezdődik elférnek egy vödörben, itt már az első két bit alapján tudjuk, hogy melyik vödörbe kerül a rekord. Hasonló a helyzet a 01 és 11 kezdetű $h(K)$ értékekkel is. Egyedül az 10 első két bit esetén számít, hogy mi a harmadik (az 10 kezdetűek nem férnek el egy vödörben).

Műveletek megvalósítása

- **keresés**: az adott K keresési kulcsra kiszámoljuk $h(K)$ első d bitjét, a vödörkatalógus megfelelő mutatója alapján tudjuk, hogy honnan kell beolvasni az egy blokkból álló vödröt, amiben a rekordnak lennie kell

Műveletek megvalósítása

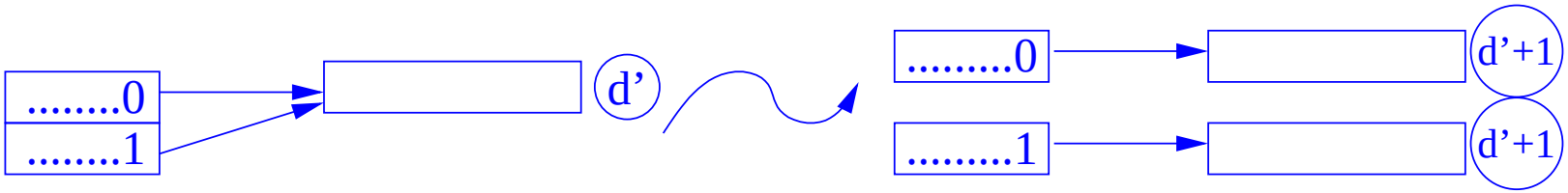
- **keresés**: az adott K keresési kulcsra kiszámoljuk $h(K)$ első d bitjét, a vödörkatalógus megfelelő mutatója alapján tudjuk, hogy honnan kell beolvasni az egy blokkból álló vödröt, amiben a rekordnak lennie kell
- **beszúrás**: beszúrni kívánt rekord K értékére kiszámoljuk $h(K)$-t, vödörkatalógus mutatóját követve behozzuk a blokkot, ahova kerülnie kell. Ha belefér ez a rekord is, akkor belerakjuk és kiírjuk a blokkot.

Műveletek megvalósítása

- **keresés**: az adott K keresési kulcsra kiszámoljuk $h(K)$ első d bitjét, a vödörkatalógus megfelelő mutatója alapján tudjuk, hogy honnan kell beolvasni az egy blokkból álló vödröt, amiben a rekordnak lennie kell
- **beszúrás**: beszúrni kívánt rekord K értékére kiszámoljuk $h(K)$-t, vödörkatalógus mutatóját követve behozzuk a blokkot, ahova kerülnie kell. Ha belefér ez a rekord is, akkor belerakjuk és kiírjuk a blokkot.
  Ha nem fér bele, akkor szét kell szedni a blokkot két részre, egy vödör helyett lesz kettő. Két eset van:

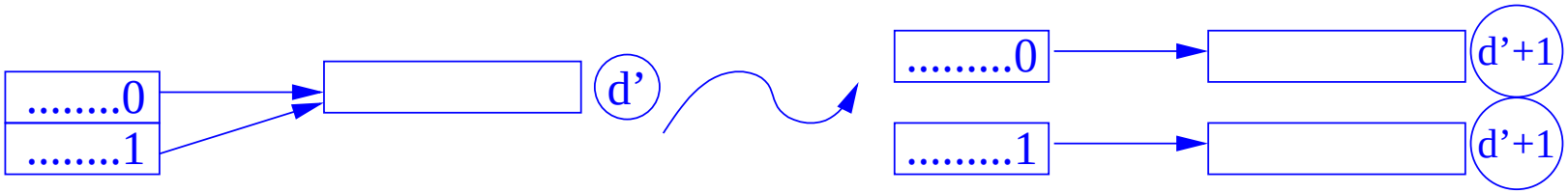
Műveletek megvalósítása

- **keresés**: az adott K keresési kulcsra kiszámoljuk $h(K)$ első d bitjét, a vödörkatalógus megfelelő mutatója alapján tudjuk, hogy honnan kell beolvasni az egy blokkból álló vödröt, amiben a rekordnak lennie kell
- **beszúrás**: beszúrni kívánt rekord K értékére kiszámoljuk $h(K)$-t, vödörkatalógus mutatóját követve behozzuk a blokkot, ahova kerülnie kell. Ha belefér ez a rekord is, akkor belerakjuk és kiírjuk a blokkot.
  Ha nem fér bele, akkor szét kell szedni a blokkot két részre, egy vödör helyett lesz kettő. Két eset van:
  - Ha $d' < d$, a vödör lokális mélysége kisebb d-nél: $d' := d' + 1 \leq d$ lesz, a vödörkatalógus nem változik, csak az eddig ugyanarra a vödörre mutató két mutató most majd két külön blokkra mutat:



Műveletek megvalósítása

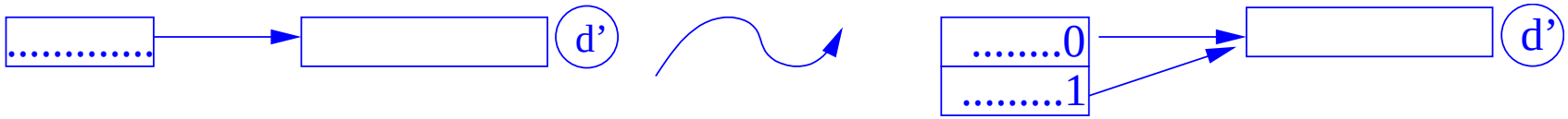
- **keresés**: az adott K keresési kulcsra kiszámoljuk $h(K)$ első d bitjét, a vödörkatalógus megfelelő mutatója alapján tudjuk, hogy honnan kell beolvasni az egy blokkból álló vödröt, amiben a rekordnak lennie kell
- **beszúrás**: beszúrni kívánt rekord K értékére kiszámoljuk $h(K)$-t, vödörkatalógus mutatóját követve behozzuk a blokkot, ahova kerülnie kell. Ha belefér ez a rekord is, akkor belerakjuk és kiírjuk a blokkot.
  Ha nem fér bele, akkor szét kell szedni a blokkot két részre, egy vödör helyett lesz kettő. Két eset van:
  - Ha $d' < d$, a vödör lokális mélysége kisebb d-nél: $d' := d' + 1 \leq d$ lesz, a vödörkatalógus nem változik, csak az eddig ugyanarra a vödörre mutató két mutató most majd két külön blokkra mutat:



Természetesen az eddig egy vödörben levő rekordokat szét kell válogatni aszerint, hogy a $d' + 1$-edik bitje mi a $h(K)$-nak

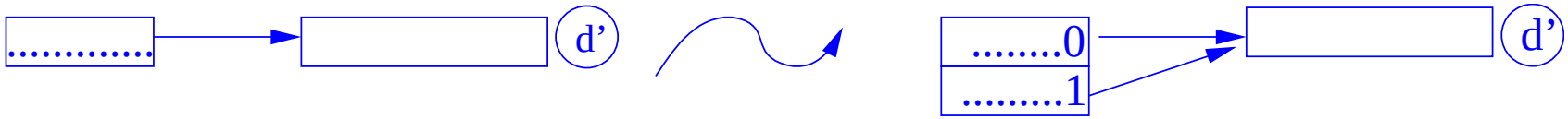
Műveletek megvalósítása

- Ha $d' = d$, azaz már nem lehet növelni a lokális mélységet d változtatása nélkül: $d := d + 1$, a vödörkatalógust megduplázom egy $x_1 \ldots x_d$ bejegyzés helyett lesz kettő: $x_1 \ldots x_d$ 0 és $x_1 \ldots x_d$ 1, az ezekből kiinduló két mutató ugyanoda megy, ahova a korábbi egy, a lokális mélység nem változik.



Műveletek megvalósítása

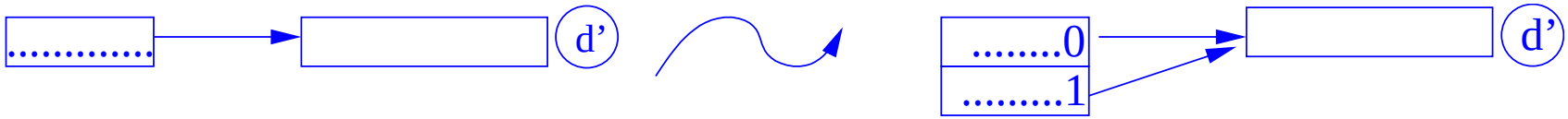
- Ha $d' = d$, azaz már nem lehet növelni a lokális mélységet d változtatása nélkül: $d := d + 1$, a vödörkatalógust megduplázom egy $x_1 \ldots x_d$ bejegyzés helyett lesz kettő: $x_1 \ldots x_d$ 0 és $x_1 \ldots x_d$ 1, az ezekből kiinduló két mutató ugyanoda megy, ahova a korábbi egy, a lokális mélység nem változik.



Ezek után már végrehajtható a vödörszétvágás úgy, mint az előbb, mert most már $d' < d$ mindenhol az új d-vel.

Műveletek megvalósítása

- Ha $d' = d$, azaz már nem lehet növelni a lokális mélységet d változtatása nélkül: $d := d + 1$, a vödörkatalógust megduplázom egy $x_1 \ldots x_d$ bejegyzés helyett lesz kettő: $x_1 \ldots x_d$ 0 és $x_1 \ldots x_d$ 1, az ezekből kiinduló két mutató ugyanoda megy, ahova a korábbi egy, a lokális mélység nem változik.

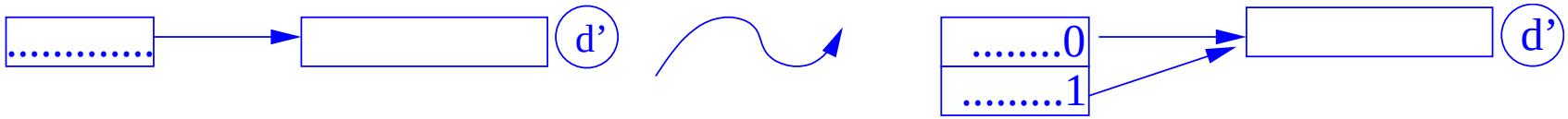


Ezek után már végrehajtható a vödörszétvágás úgy, mint az előbb, mert most már $d' < d$ mindenhol az új d-vel.

- **törlés**: keresés, aztán pedig törlés a vödörből;

Műveletek megvalósítása

- Ha $d' = d$, azaz már nem lehet növelni a lokális mélységet d változtatása nélkül: $d := d + 1$, a vödörkatalógust megduplázom egy $x_1 \ldots x_d$ bejegyzés helyett lesz kettő: $x_1 \ldots x_d$ 0 és $x_1 \ldots x_d$ 1, az ezekből kiinduló két mutató ugyanoda megy, ahova a korábbi egy, a lokális mélység nem változik.

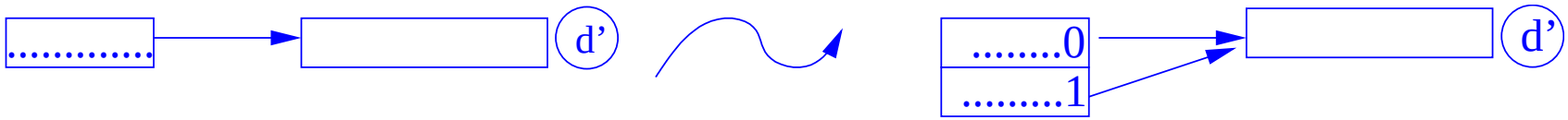


Ezek után már végrehajtható a vödörszétvágás úgy, mint az előbb, mert most már $d' < d$ mindenhol az új d-vel.

- **törlés**: keresés, aztán pedig törlés a vödörből; ha ettől olyan helyzet áll elő, hogy a vödör összevonható a párjával (ahol az első $d' - 1$ bit egyezik, de a d'-edik más), akkor összevonás (két mutató ugyanoda fog mutatni) és d' csökkentése eggyel.

Műveletek megvalósítása

- Ha $d' = d$, azaz már nem lehet növelni a lokális mélységet d változtatása nélkül: $d := d + 1$, a vödörkatalógust megduplázom egy $x_1 \ldots x_d$ bejegyzés helyett lesz kettő: $x_1 \ldots x_d$ 0 és $x_1 \ldots x_d$ 1, az ezekből kiinduló két mutató ugyanoda megy, ahova a korábbi egy, a lokális mélység nem változik.

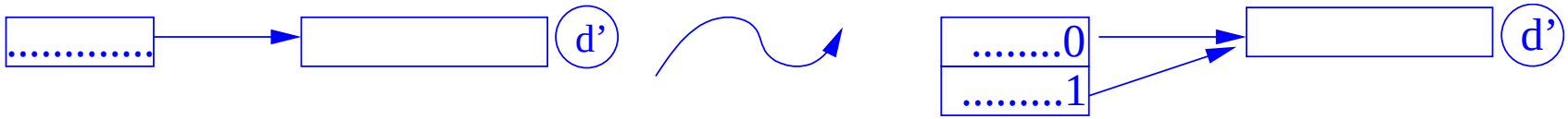


Ezek után már végrehajtható a vödörszétvágás úgy, mint az előbb, mert most már $d' < d$ mindenhol az új d-vel.

- **törlés**: keresés, aztán pedig törlés a vödörből; ha ettől olyan helyzet áll elő, hogy a vödör összevonható a párjával (ahol az első $d' - 1$ bit egyezik, de a d'-edik más), akkor összevonás (két mutató ugyanoda fog mutatni) és d' csökkentése eggyel. Ha minden $d' < d$, akkor d-t is csökkentjük eggyel.

Műveletek megvalósítása

- Ha $d' = d$, azaz már nem lehet növelni a lokális mélységet d változtatása nélkül: $d := d + 1$, a vödörkatalógust megduplázom egy $x_1 \ldots x_d$ bejegyzés helyett lesz kettő: $x_1 \ldots x_d$ 0 és $x_1 \ldots x_d$ 1, az ezekből kiinduló két mutató ugyanoda megy, ahova a korábbi egy, a lokális mélység nem változik.



Ezek után már végrehajtható a vödörszétvágás úgy, mint az előbb, mert most már $d' < d$ mindenhol az új d-vel.

- **törlés**: keresés, aztán pedig törlés a vödörből; ha ettől olyan helyzet áll elő, hogy a vödör összevonható a párjával (ahol az első $d' - 1$ bit egyezik, de a d'-edik más), akkor összevonás (két mutató ugyanoda fog mutatni) és d' csökkentése eggyel. Ha minden $d' < d$, akkor d-t is csökkentjük eggyel.

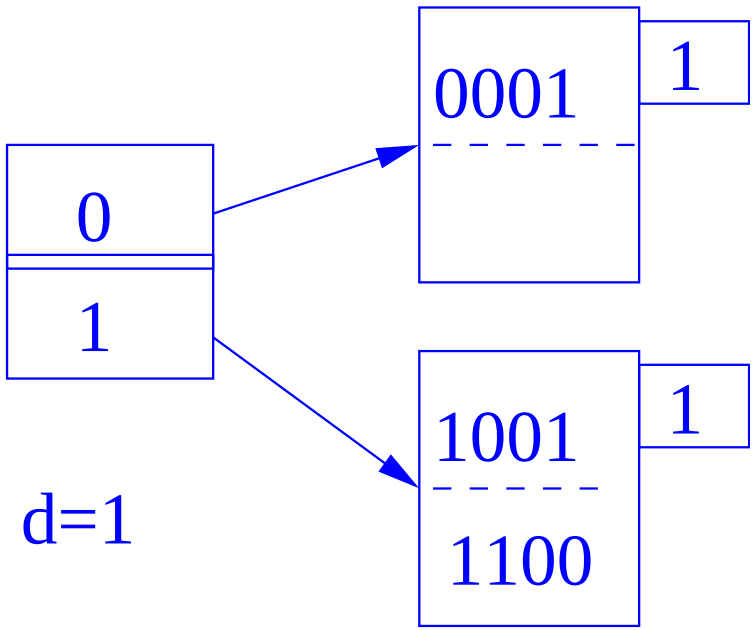
A fentiek miatt csak akkor mutathat két mutató ugyanarra a d' lokális mélységű vödörre, ha a két megfelelő érték első d' bitje megegyezik.

Példa

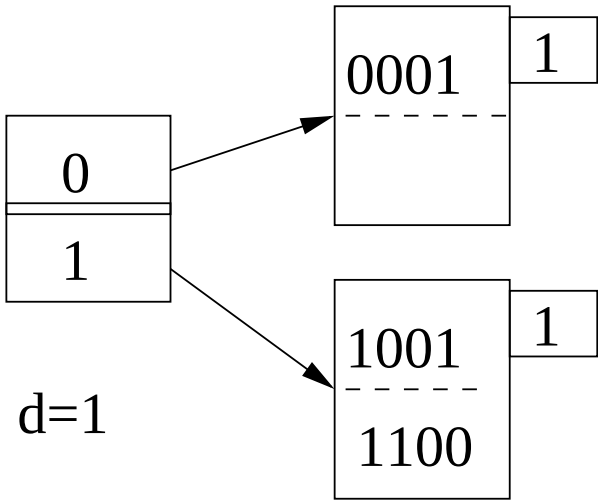
Növelhető hash segítségével akarjuk tárolni az adatainkat. Feltesszük, hogy egy lapra két rekord fér. A hash függvény 4 bites számot ad vissza, de jelenleg még csak egy bitet használunk ($d = 1$), mivel eddig csak három elem (0001, 1001, 1100) van a táblázatban (az egyszerűség kedvéért az elemek helyett azt az értéket irjuk be, amit a hash függvény visszaad).

Példa

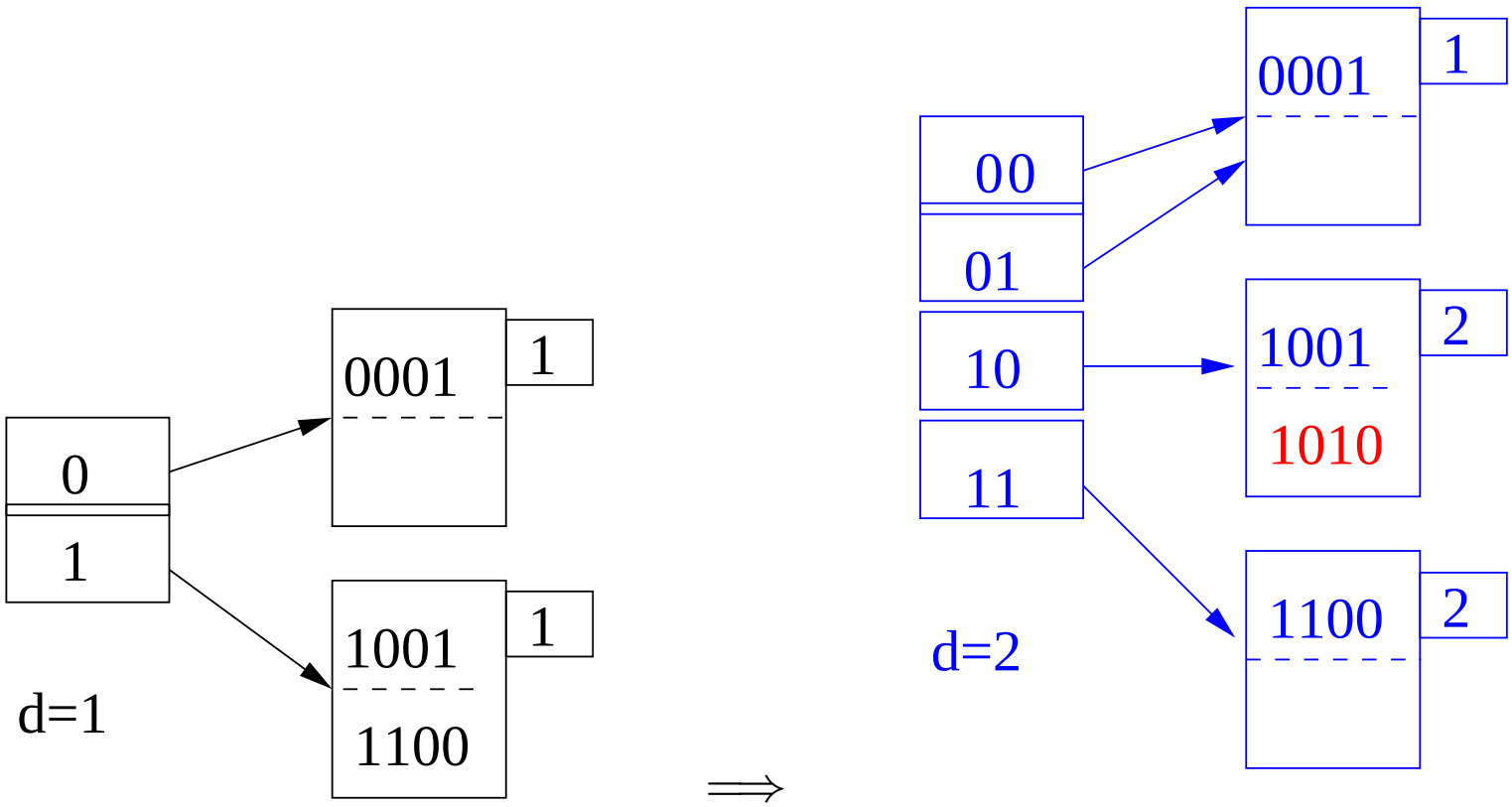
Növelhető hash segítségével akarjuk tárolni az adatainkat. Feltesszük, hogy egy lapra két rekord fér. A hash függvény 4 bites számot ad vissza, de jelenleg még csak egy bitet használunk ($d = 1$), mivel eddig csak három elem (0001, 1001, 1100) van a táblázatban (az egyszerűség kedvéért az elemek helyett azt az értéket irjuk be, amit a hash függvény visszaad).
Tehát most így néz ki a tábla, van egy bejegyzés a 0-hoz és egy az 1-hez:



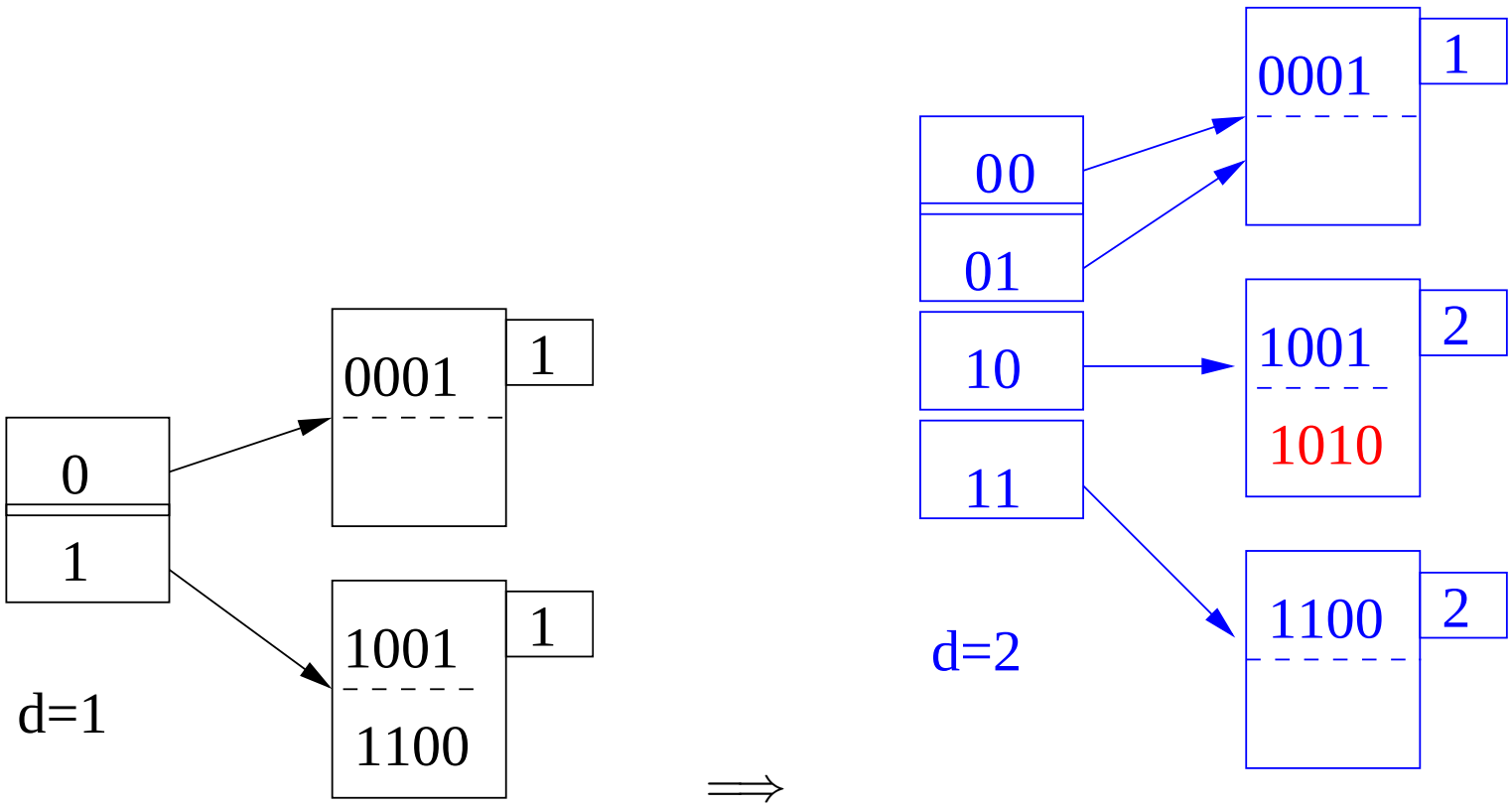
Ha most beszúrni szeretnénk egy olyan elemet, melyre a hash függvény az 1010 értéket adja, akkor az új tábla ilyen lesz:



Ha most beszúrni szeretnénk egy olyan elemet, melyre a hash függvény az 1010 értéket adja, akkor az új tábla ilyen lesz:

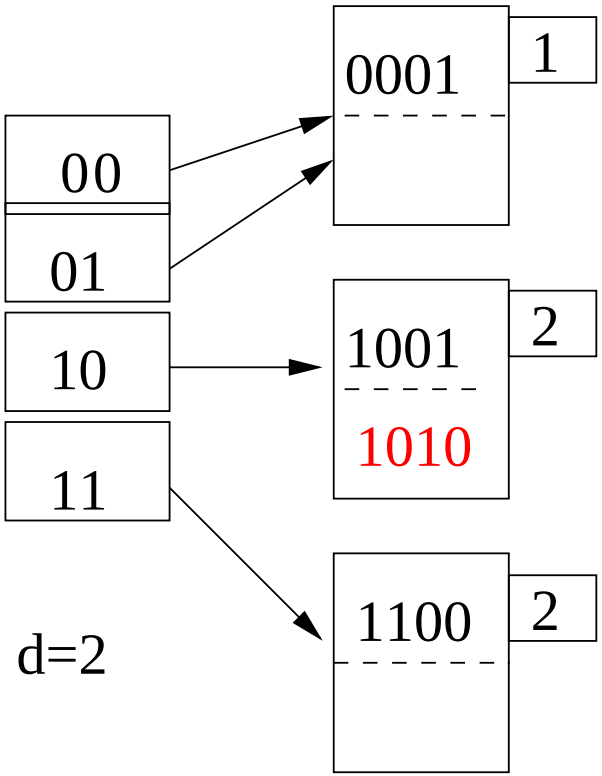


Ha most beszúrni szeretnénk egy olyan elemet, melyre a hash függvény az 1010 értéket adja, akkor az új tábla ilyen lesz:

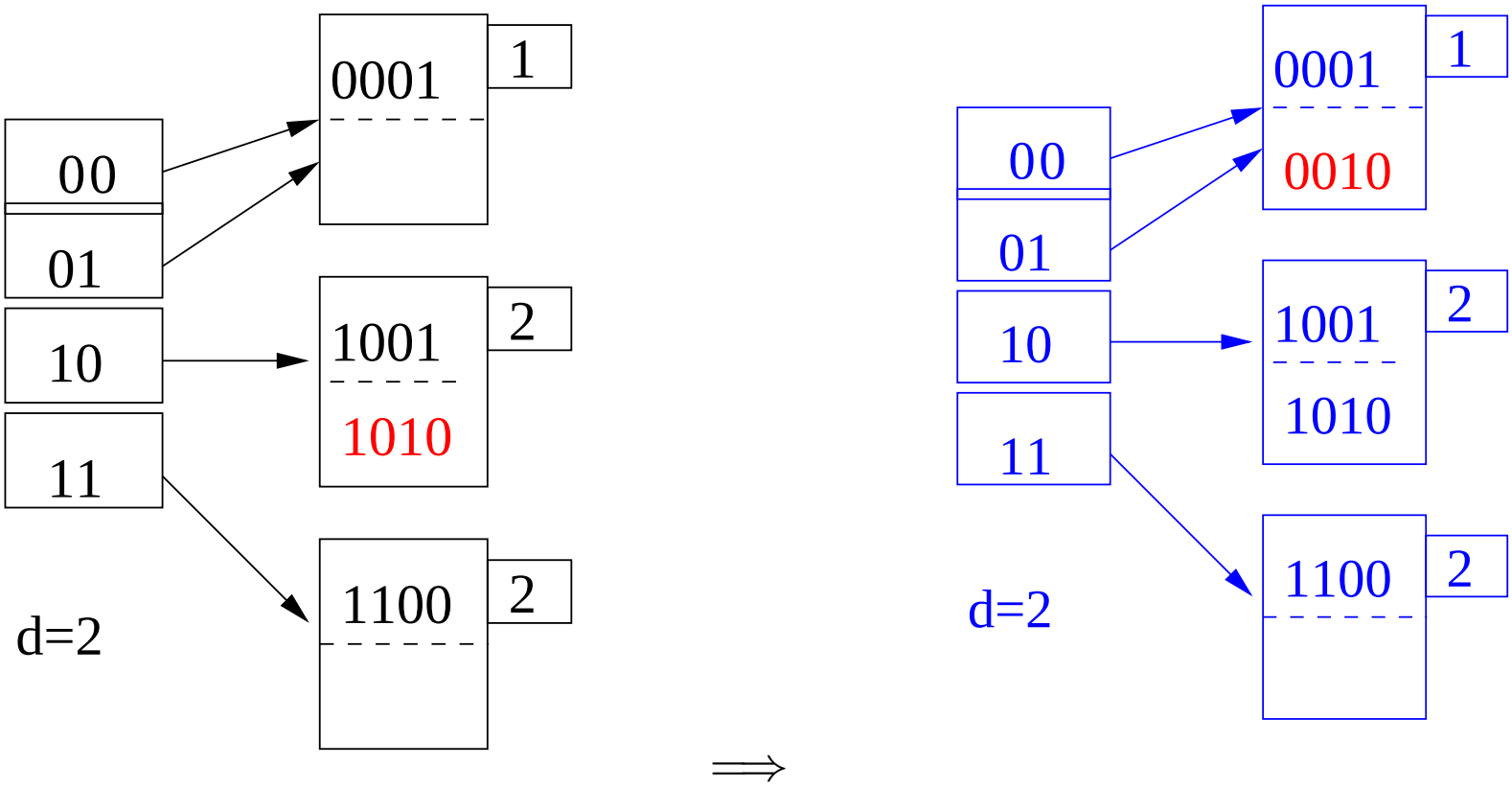


Mivel az 1-hez tartozó lap már betelt, ezért itt a lokális mélységet növelni kellett 1-ről 2-re (két bitet akarunk figyelembe venni), de így a *d* értékét is növelnünk kellett.

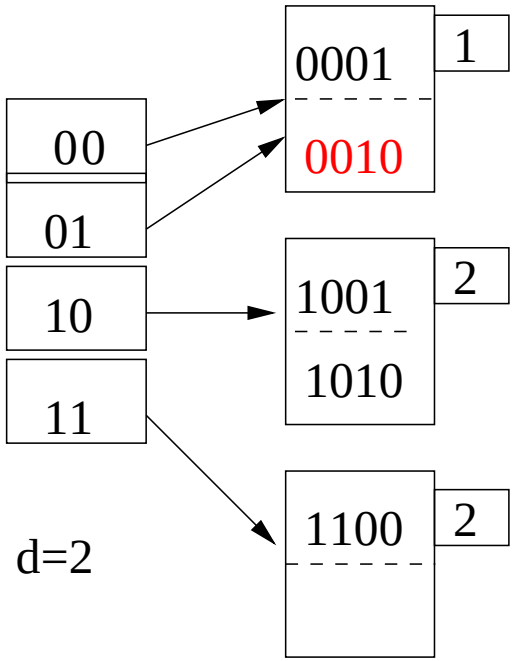
Ha most jön egy 0010 hash-értékű elem, akkor azt simán be tudjuk rakni a helyére:



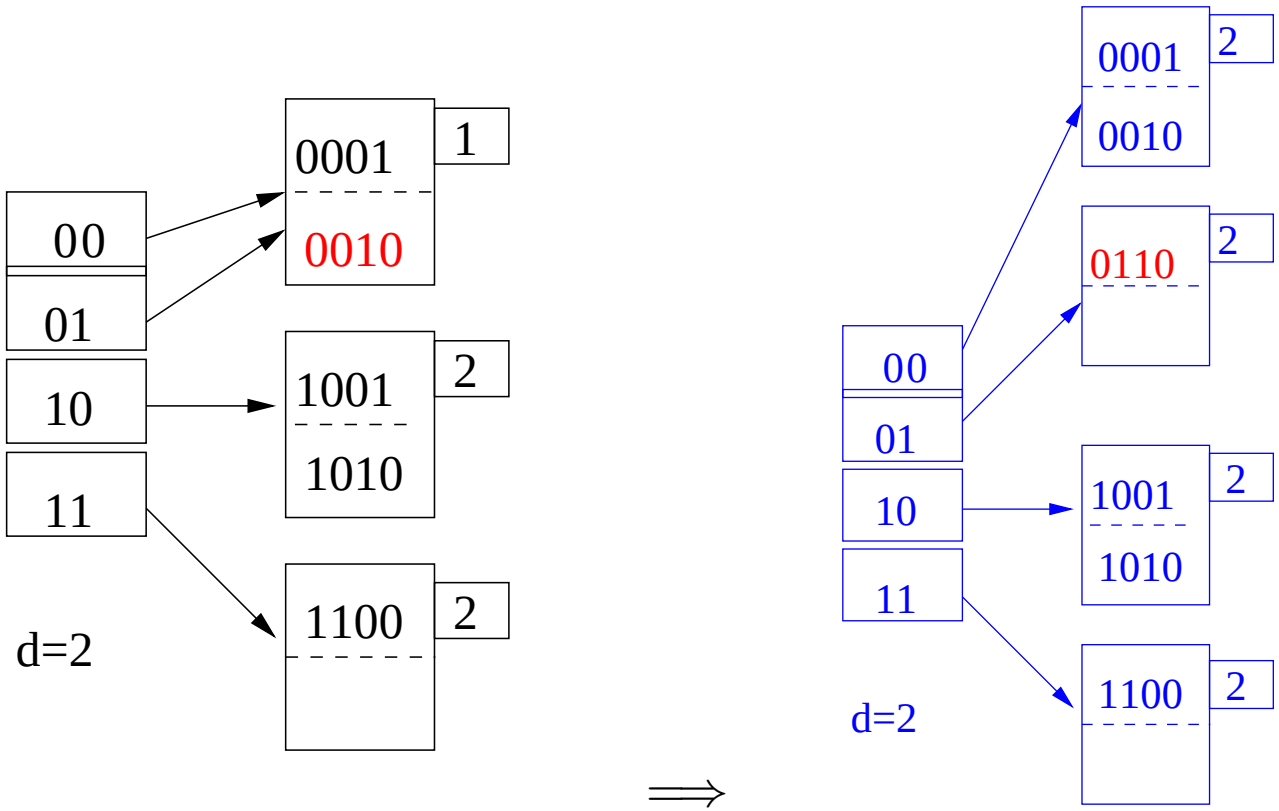
Ha most jön egy 0010 hash-értékű elem, akkor azt simán be tudjuk rakni a helyére:



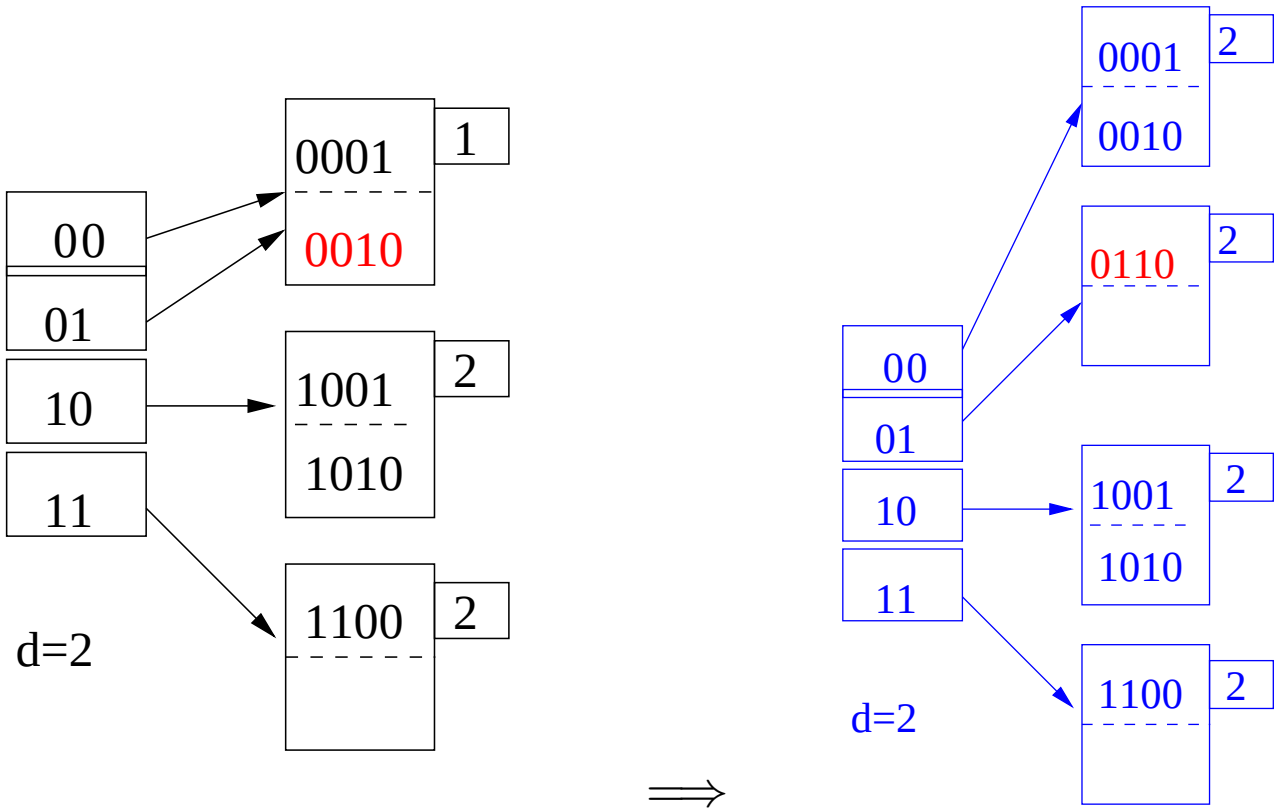
De ha ezután olyan jön, ahol a hash függvény értéke 0110, akkor új lapot kell létrehoznunk, de a *d* értékét nem kell növelnünk, mert elég lesz az eddig ugyanoda mutató két mutatót szétszedni és ezen új lapoknál a lokális mélységet 2-re állítani.



De ha ezután olyan jön, ahol a hash függvény értéke 0110, akkor új lapot kell létrehoznunk, de a *d* értékét nem kell növelnünk, mert elég lesz az eddig ugyanoda mutató két mutatót szétszedni és ezen új lapoknál a lokális mélységet 2-re állítani.



De ha ezután olyan jön, ahol a hash függvény értéke 0110, akkor új lapot kell létrehoznunk, de a *d* értékét nem kell növelnünk, mert elég lesz az eddig ugyanoda mutató két mutatót szétszedni és ezen új lapoknál a lokális mélységet 2-re állítani.



Ha ezután jönne egy olyan elem, aminek az értéke 00-val vagy 10-val kezdődik, akkor újabb lapra lenne szükségünk, de ehhez már a *d* értékét is növelni kellene.

A dinamikus hash értékelése

Előnyök:

- egy I/O művelettel minden megvan, ha a vödörkatalógus a belső memóriában van
- a struktúra igazodik az állomány növekedéséhez

A dinamikus hash értékelése

Előnyök:

- egy I/O művelettel minden megvan, ha a vödörkatalógus a belső memóriában van
- a struktúra igazodik az állomány növekedéséhez

Bajok:

- szerencsétlen esetben nagyon nagy vödörkatalógust kell építeni kevés rekord miatt (legrosszabb esetben lényegében a szekvenciális keresést kapjuk vissza)

A dinamikus hash értékelése

Előnyök:

- egy I/O művelettel minden megvan, ha a vödörkatalógus a belső memóriában van
- a struktúra igazodik az állomány növekedéséhez

Bajok:

- szerencsétlen esetben nagyon nagy vödörkatalógust kell építeni kevés rekord miatt (legrosszabb esetben lényegében a szekvenciális keresést kapjuk vissza)
- a $h(K)$ értéket se lehet a végtelenségig továbbszámolni: ha $h(K)$ már teljes hosszáig kiszámolandó, akkor nem lehet tovább növelni a struktúrát

Adatbázisok elmélete

Indexek

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

20. előadás

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Motiváló ötlet:

- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.
Motiváló ötlet:

- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)
- ha a polcokon ábécé szerint vannak sorban a könyvek, akkor a polcok elején álló könyvek megvizsgálásával gyorsan eldönthető, hogy melyiken kell keresni (ritka indexre motiváció)

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Motiváló ötlet:

- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)
- ha a polcokon ábécé szerint vannak sorban a könyvek, akkor a polcok elején álló könyvek megvizsgálásával gyorsan eldönthető, hogy melyiken kell keresni (ritka indexre motiváció)

Indexek általános jellemzői:

- rendezettséget figyelembe veszi (ellentétben a hash-sel)

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Motiváló ötlet:

- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)
- ha a polcokon ábécé szerint vannak sorban a könyvek, akkor a polcok elején álló könyvek megvizsgálásával gyorsan eldönthető, hogy melyiken kell keresni (ritka indexre motiváció)

Indexek általános jellemzői:

- rendezettséget figyelembe veszi (ellentétben a hash-sel)
- van korlát a legrosszabb esetre is (ellentétben a hash-el, ami viszont átlagosan jobb)

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Motiváló ötlet:

- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)
- ha a polcokon ábécé szerint vannak sorban a könyvek, akkor a polcok elején álló könyvek megvizsgálásával gyorsan eldönthető, hogy melyiken kell keresni (ritka indexre motiváció)

Indexek általános jellemzői:

- rendezettséget figyelembe veszi (ellentétben a hash-sel)
- van korlát a legrosszabb esetre is (ellentétben a hash-el, ami viszont átlagosan jobb)
- jól bírja a dinamikus bővülést (ellentétben a hash-el)

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Motiváló ötlet:

- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)
- ha a polcokon ábécé szerint vannak sorban a könyvek, akkor a polcok elején álló könyvek megvizsgálásával gyorsan eldönthető, hogy melyiken kell keresni (ritka indexre motiváció)

Indexek általános jellemzői:

- rendezettséget figyelembe veszi (ellentétben a hash-sel)
- van korlát a legrosszabb esetre is (ellentétben a hash-el, ami viszont átlagosan jobb)
- jól bírja a dinamikus bővülést (ellentétben a hash-el)
- kulcs rendezett halmazból kerül ki (de nem féltétlenül egyedi)

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Motiváló ötlet:

- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)
- ha a polcokon ábécé szerint vannak sorban a könyvek, akkor a polcok elején álló könyvek megvizsgálásával gyorsan eldönthető, hogy melyiken kell keresni (ritka indexre motiváció)

Indexek általános jellemzői:

- rendezettséget figyelembe veszi (ellentétben a hash-sel)
- van korlát a legrosszabb esetre is (ellentétben a hash-el, ami viszont átlagosan jobb)
- jól bírja a dinamikus bővülést (ellentétben a hash-el)
- kulcs rendezett halmazból kerül ki (de nem féltétlenül egyedi)
- lehet ugyanarra az állományra több index is készítve

Az indexstruktúra vázlatos felépítése

Az indexállományban vannak a keresést segítő infók, a főállományban vannak a tárolt adatok, a rekordok.

Az indexstruktúra vázlatos felépítése

Az indexállományban vannak a keresést segítő infók, a főállományban vannak a tárolt adatok, a rekordok.
A főállomány lehet esetleg rendezett a keresési kulcs szerint, de nem feltétlenül az.

Az indexstruktúra vázlatos felépítése

Az indexállományban vannak a keresést segítő infók, a főállományban vannak a tárolt adatok, a rekordok.
A főállomány lehet esetleg rendezett a keresési kulcs szerint, de nem feltétlenül az.
Az indexállományban indexrekordok vannak, ezek felépítése:

kulcs	mutató

A kulcs valamelyik főállománybeli rekordhoz tartozó kulcsérték, a mutató pedig a főállományba mutat, oda, ahol ez a rekord van.

Az indexstruktúra vázlatos felépítése

Az indexállományban vannak a keresést segítő infók, a főállományban vannak a tárolt adatok, a rekordok.
A főállomány lehet esetleg rendezett a keresési kulcs szerint, de nem feltétlenül az.
Az indexállományban indexrekordok vannak, ezek felépítése:

A kulcs valamelyik főállománybeli rekordhoz tartozó kulcsérték, a mutató pedig a főállományba mutat, oda, ahol ez a rekord van.
Aszerint, hogy minden főállománybeli rekordra van rá mutató indexbejegyzés vagy pedig csak néhányra, sűrű illetve ritka indexről beszélhetünk. (Majd látjuk, hogy ez két nagyon más helyzet lesz.)

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.
Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.
Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés
- ritka index, azaz nem minden főállománybeli rekordra van indexbejegyzés, csak blokkonként egyre

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.
Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés
- ritka index, azaz nem minden főállománybeli rekordra van indexbejegyzés, csak blokkonként egyre
- a főállomány vagy rendezett a keresési kulcs szerint vagy lényegében rendezett (blokkokon belül esetleg nem rendezettek a rekordok, de a blokkok egymáshoz képest rendezetten vannak)

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.
Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés
- ritka index, azaz nem minden főállománybeli rekordra van indexbejegyzés, csak blokkonként egyre
- a főállomány vagy rendezett a keresési kulcs szerint vagy lényegében rendezett (blokkokon belül esetleg nem rendezettek a rekordok, de a blokkok egymáshoz képest rendezetten vannak)
- az indexállomány rendezett a keresési kulcs szerint (az indexállomány is háttértéron van)

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.
Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés
- ritka index, azaz nem minden főállománybeli rekordra van indexbejegyzés, csak blokkonként egyre
- a főállomány vagy rendezett a keresési kulcs szerint vagy lényegében rendezett (blokkokon belül esetleg nem rendezettek a rekordok, de a blokkok egymáshoz képest rendezetten vannak)
- az indexállomány rendezett a keresési kulcs szerint (az indexállomány is háttértéron van)
- az indexbejegyzésben szereplő kulcs a rendezett esetben a blokk legelső kulcsa, a lényegében rendezett esetben a blokk legkisebb kulcsa

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.
Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés
- ritka index, azaz nem minden főállománybeli rekordra van indexbejegyzés, csak blokkonként egyre
- a főállomány vagy rendezett a keresési kulcs szerint vagy lényegében rendezett (blokkokon belül esetleg nem rendezettek a rekordok, de a blokkok egymáshoz képest rendezetten vannak)
- az indexállomány rendezett a keresési kulcs szerint (az indexállomány is háttértéron van)
- az indexbejegyzésben szereplő kulcs a rendezett esetben a blokk legelső kulcsa, a lényegében rendezett esetben a blokk legkisebb kulcsa
- a mutató az indexbejegyzésben arra a blokkra mutat, ahol a bejegyzésben szereplő kulcshoz tartozó rekord van

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.
Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés
- ritka index, azaz nem minden főállománybeli rekordra van indexbejegyzés, csak blokkonként egyre
- a főállomány vagy rendezett a keresési kulcs szerint vagy lényegében rendezett (blokkokon belül esetleg nem rendezettek a rekordok, de a blokkok egymáshoz képest rendezetten vannak)
- az indexállomány rendezett a keresési kulcs szerint (az indexállomány is háttértéron van)
- az indexbejegyzésben szereplő kulcs a rendezett esetben a blokk legelső kulcsa, a lényegében rendezett esetben a blokk legkisebb kulcsa
- a mutató az indexbejegyzésben arra a blokkra mutat, ahol a bejegyzésben szereplő kulcshoz tartozó rekord van

A fentiek miatt a ritka index kivonata lesz a főállománynak, tartalmazza rendezetten a blokkok legkisebb kulcsait.

Keresés

Adott a keresési kulcs értéke, meg kell találni az ilyen rekordokat.

Keresés

Adott a keresési kulcs értéke, meg kell találni az ilyen rekordokat.

1. Először az indexállományban megkeressük azt a legnagyobb indexbejegyzést, aminek kulcsa még éppen nem nagyobb, mint az adott keresési érték.

Keresés

Adott a keresési kulcs értéke, meg kell találni az ilyen rekordokat.

1. Először az indexállományban megkeressük azt a legnagyobb indexbejegyzést, aminek kulcsa még éppen nem nagyobb, mint az adott keresési érték. Ez átlagosan $\frac{n}{2}$ I/O művelet, ha n blokkból áll az indexállomány.

Keresés

Adott a keresési kulcs értéke, meg kell találni az ilyen rekordokat.

1. Először az indexállományban megkeressük azt a legnagyobb indexbejegyzést, aminek kulcsa még éppen nem nagyobb, mint az adott keresési érték. Ez átlagosan $\frac{n}{2}$ I/O művelet, ha n blokkból áll az indexállomány.
2. Az előbb megtalált indexbejegyzéshez tartozó blokkban (plusz egy I/O művelet) megkeressük a rekordunkat.

Keresés

Adott a keresési kulcs értéke, meg kell találni az ilyen rekordokat.

1. Először az indexállományban megkeressük azt a legnagyobb indexbejegyzést, aminek kulcsa még éppen nem nagyobb, mint az adott keresési érték. Ez átlagosan $\frac{n}{2}$ I/O művelet, ha n blokkból áll az indexállomány.
2. Az előbb megtalált indexbejegyzéshez tartozó blokkban (plusz egy I/O művelet) megkeressük a rekordunkat.

Megjegyzések:

1. Ha van valami plusz struktúra, ami segíti a gyors keresést az indexállományban, akkor jobbat is lehet, mint $\frac{n}{2}$.

Keresés

Adott a keresési kulcs értéke, meg kell találni az ilyen rekordokat.

1. Először az indexállományban megkeressük azt a legnagyobb indexbejegyzést, aminek kulcsa még éppen nem nagyobb, mint az adott keresési érték. Ez átlagosan $\frac{n}{2}$ I/O művelet, ha n blokkból áll az indexállomány.
2. Az előbb megtalált indexbejegyzéshez tartozó blokkban (plusz egy I/O művelet) megkeressük a rekordunkat.

Megjegyzések:

1. Ha van valami plusz struktúra, ami segíti a gyors keresést az indexállományban, akkor jobbat is lehet, mint $\frac{n}{2}$.
2. A blokkon belül, a főállományban már bárhogy kereshetünk, de leginkább szekvenciálisan megy.

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük;

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova;

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból
- **törlés**: hasonlóan, mint a beszúrás, először keresés, aztán törlés a blokkból;

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból
- **törlés**: hasonlóan, mint a beszúrás, először keresés, aztán törlés a blokkból; ha épp a minimális rekordot töröltem a blokkból $\Longrightarrow$ indexbejegyzés módosítása

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból
- **törlés**: hasonlóan, mint a beszúrás, először keresés, aztán törlés a blokkból;
  ha épp a minimális rekordot töröltem a blokkból $\Longrightarrow$ indexbejegyzés módosítása
  ha nagyon ritkák a blokkok $\Longrightarrow$ esetleg lapösszevonás (és persze az indexállomány módosítása is ilyenkor), de általában nem érdemes

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból
- **törlés**: hasonlóan, mint a beszúrás, először keresés, aztán törlés a blokkból; ha épp a minimális rekordot töröltem a blokkból $\Longrightarrow$ indexbejegyzés módosítása ha nagyon ritkák a blokkok $\Longrightarrow$ esetleg lapösszevonás (és persze az indexállomány módosítása is ilyenkor), de általában nem érdemes
- **tól-ig**: valahonnan valameddig terjedő értékű rekordok keresése: a „tól” érték keresése, aztán a főállomány végigolvasása az „ig” értékig (a főállomány folyamatos elérése megoldott)

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból
- **törlés**: hasonlóan, mint a beszúrás, először keresés, aztán törlés a blokkból; ha épp a minimális rekordot töröltem a blokkból $\Longrightarrow$ indexbejegyzés módosítása ha nagyon ritkák a blokkok $\Longrightarrow$ esetleg lapösszevonás (és persze az indexállomány módosítása is ilyenkor), de általában nem érdemes
- **tól-ig**: valahonnan valameddig terjedő értékű rekordok keresése: a „tól” érték keresése, aztán a főállomány végigolvasása az „ig” értékig (a főállomány folyamatos elérése megoldott)
- **módosítás**: ha kulcsot nem érint: keresés, átírás;

További műveletek

- **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
  - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
  - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból
- **törlés**: hasonlóan, mint a beszúrás, először keresés, aztán törlés a blokkból; ha épp a minimális rekordot töröltem a blokkból $\Longrightarrow$ indexbejegyzés módosítása ha nagyon ritkák a blokkok $\Longrightarrow$ esetleg lapösszevonás (és persze az indexállomány módosítása is ilyenkor), de általában nem érdemes
- **tól-ig**: valahonnan valameddig terjedő értékű rekordok keresése: a „tól” érték keresése, aztán a főállomány végigolvasása az „ig” értékig (a főállomány folyamatos elérése megoldott)
- **módosítás**: ha kulcsot nem érint: keresés, átírás; ha kulcsot is érint: törlés, beszúrás

Megjegyzések:

- Tényleg használtuk, hogy a főállomány szabad (pakolgattuk a rekordokat össze-vissza). Lesz majd technika, amivel elérhető lesz, hogy a főállomány szabadnak látszódjon.

Megjegyzések:

- Tényleg használtuk, hogy a főállomány szabad (pakolgattuk a rekordokat össze-vissza). Lesz majd technika, amivel elérhető lesz, hogy a főállomány szabadnak látszódjon.
- Azt is erősen kihasználtuk, hogy (lényegében) rendezett a főállomány.

Megjegyzések:

- Tényleg használtuk, hogy a főállomány szabad (pakolgattuk a rekordokat össze-vissza). Lesz majd technika, amivel elérhető lesz, hogy a főállomány szabadnak látszódjon.
- Azt is erősen kihasználtuk, hogy (lényegében) rendezett a főállomány.
- Azért hasznos az indexállomány, mert sokkal kisebb, mint a főállomány, könnyebb benne keresni és a végen csak plusz egy I/O művelet kell a befejezéshez. De ennek ára van: karban kell tartani plusz egy struktúrát.

Többszintes ritka index

Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.

Többszintes ritka index

Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.
Ezért: többszintű index, vagyis index az indexre:

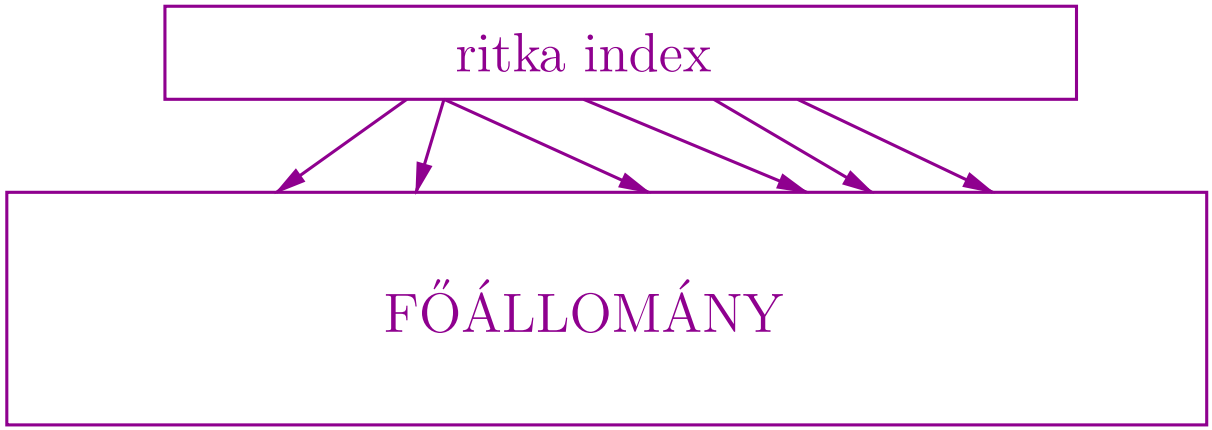
Többszintes ritka index

Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.

Ezért: többszintű index, vagyis index az indexre:

Többszintes ritka index

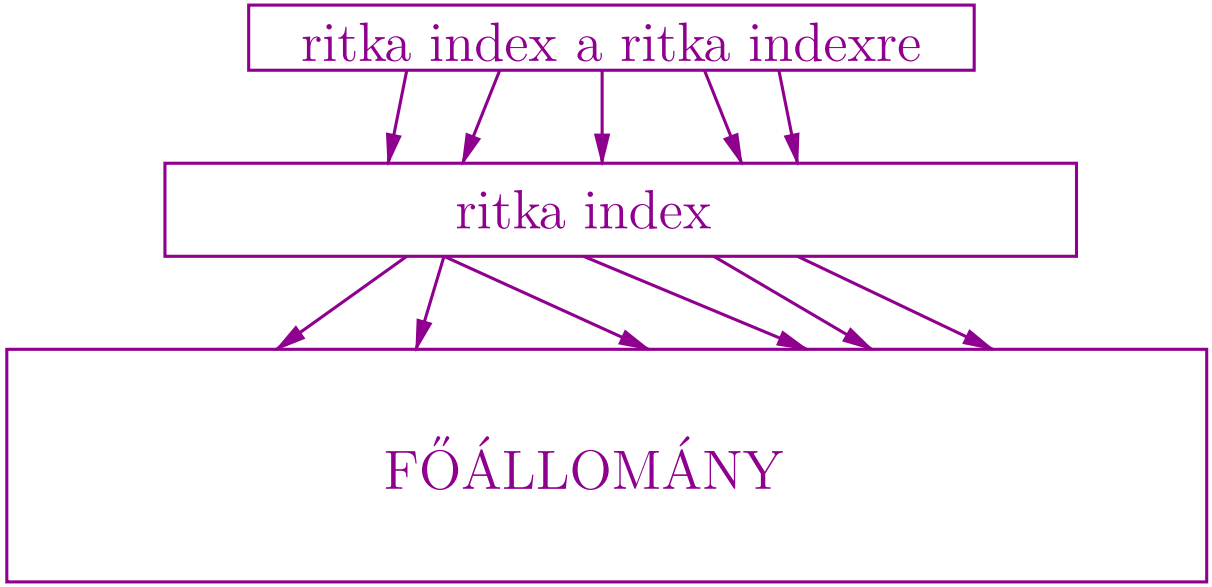
Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.
Ezért: többszintű index, vagyis index az indexre:



Többszintes ritka index

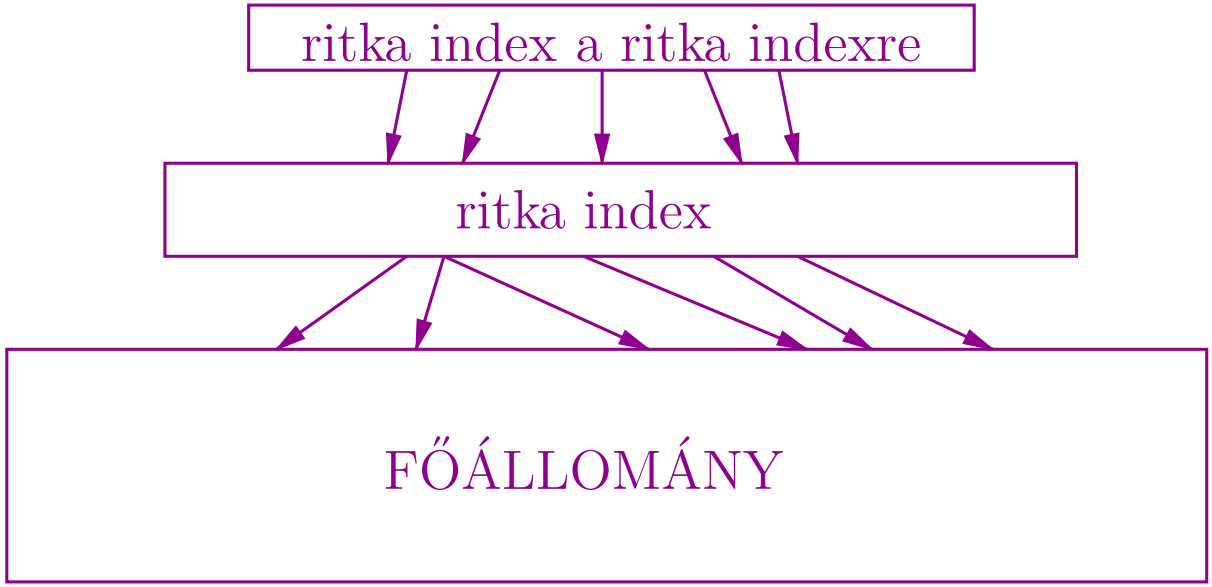
Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.

Ezért: többszintű index, vagyis index az indexre:



Többszintes ritka index

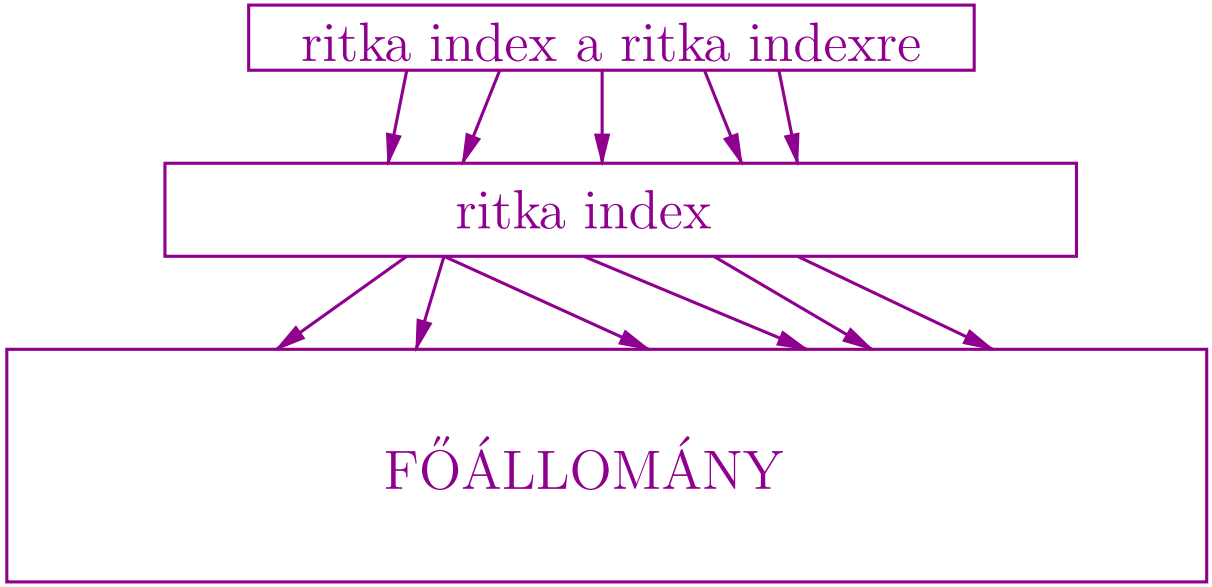
Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.
Ezért: többszintű index, vagyis index az indexre:



- a felső index még kisebb lesz, könnyebb lesz benne keresni

Többszintes ritka index

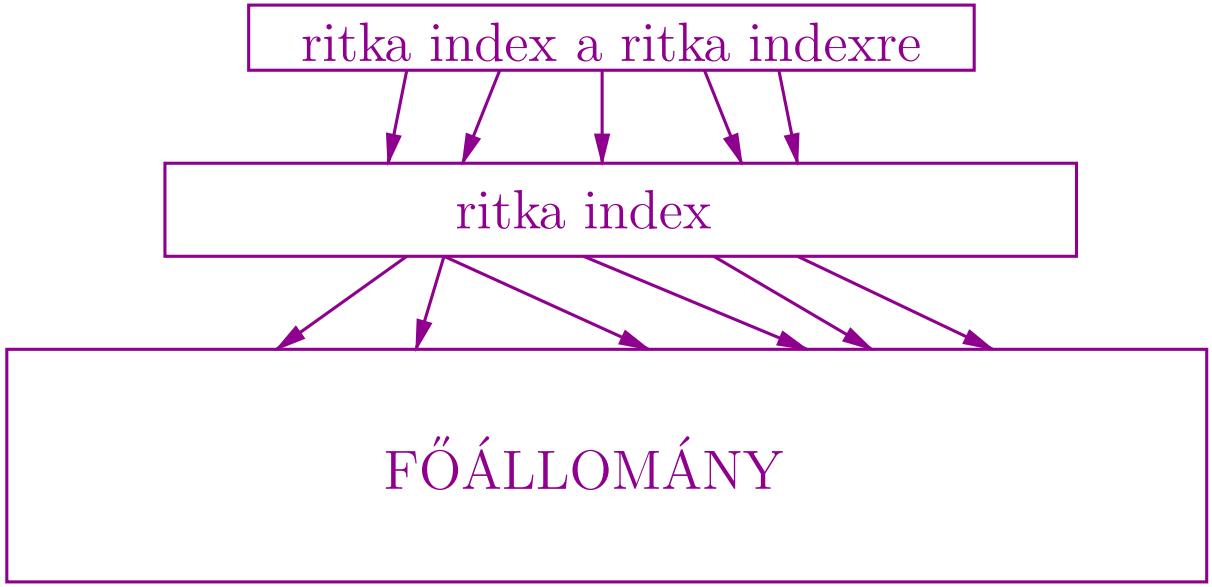
Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.
Ezért: többszintű index, vagyis index az indexre:



- a felső index még kisebb lesz, könnyebb lesz benne keresni
- a középső szint egyszerre indexe a főállománynak és „főállománya” a felső indexnek

Többszintes ritka index

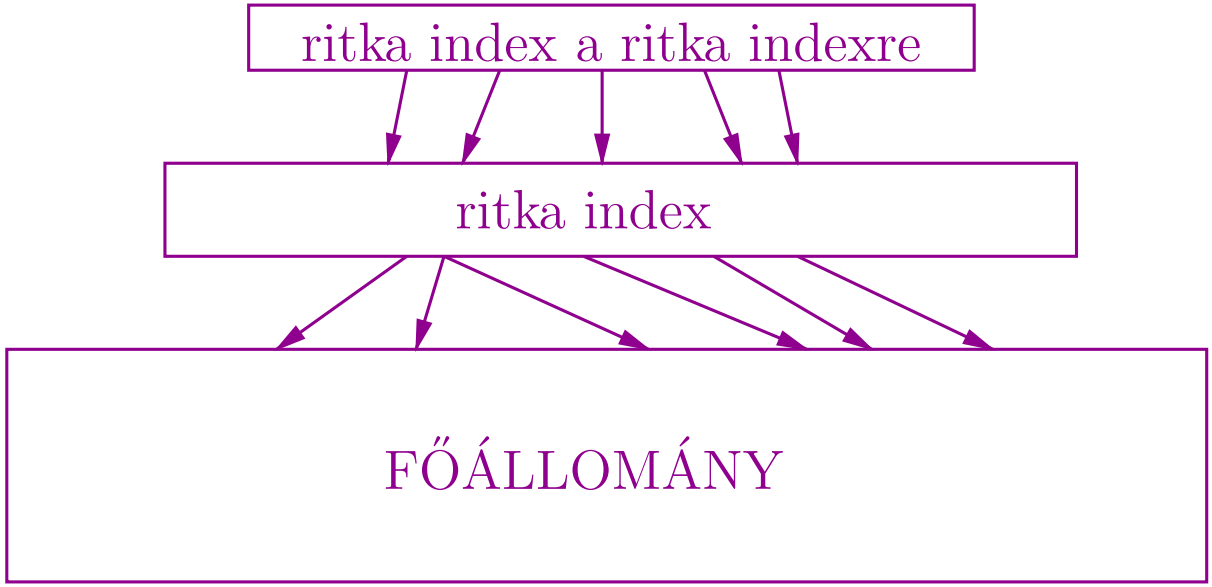
Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.
Ezért: többszintű index, vagyis index az indexre:



- a felső index még kisebb lesz, könnyebb lesz benne keresni
- a középső szint egyszerre indexe a főállománynak és „főállománya” a felső indexnek
- keresés: a legfelső szinten megkeressük a legnagyobb olyan bejegyzést, ami még kisebb a keresettnél és innen két lap beolvasásával (a megfelelő középső szintű indexlap és aztán a főállomány megfelelő lapja) megvan a keresett rekord

Többszintes ritka index

Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet.
Ezért: többszintű index, vagyis index az indexre:



- a felső index még kisebb lesz, könnyebb lesz benne keresni
- a középső szint egyszerre indexe a főállománynak és „főállománya” a felső indexnek
- keresés: a legfelső szinten megkeressük a legnagyobb olyan bejegyzést, ami még kisebb a keresettnél és innen két lap beolvasásával (a megfelelő középső szintű indexlap és aztán a főállomány megfelelő lapja) megvan a keresett rekord

Többszintes ritka index

- a többi művelet hasonlóan megy (persze, ha módosul a főállomány, akkor esetleg mindegyik indexállományt is módosítani kell)
- eggyel több szint lesz, azaz eggyel több lapelérés kell a kezdeti keresés után, de a kezdeti keresés lerövidül

Többszintes ritka index

- a többi művelet hasonlóan megy (persze, ha módosul a főállomány, akkor esetleg mindegyik indexállományt is módosítani kell)
- eggyel több szint lesz, azaz eggyel több lapelérés kell a kezdeti keresés után, de a kezdeti keresés lerövidül
- nem csak kétszintű lehet a ritka index, hanem több is

Többszintes ritka index

- a többi művelet hasonlóan megy (persze, ha módosul a főállomány, akkor esetleg mindegyik indexállományt is módosítani kell)
- eggyel több szint lesz, azaz eggyel több lapelérés kell a kezdeti keresés után, de a kezdeti keresés lerövidül
- nem csak kétszintű lehet a ritka index, hanem több is $\Longrightarrow$ dinamikusan is változhat $\Longrightarrow$ B-fa

B-fa

A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa:

- a fa levelei: a főállomány blokkjai

B-fa

A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa:

- a fa levelei: a főállomány blokkjai
- a főállomány (a levelek) rendezett a keresési kulcs szerint

B-fa

A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa:

- a fa levelei: a főállomány blokkjai
- a főállomány (a levelek) rendezett a keresési kulcs szerint
- minden levél ugyanolyan messze van a gyökértől

B-fa

A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa:

- a fa levelei: a főállomány blokkjai
- a főállomány (a levelek) rendezett a keresési kulcs szerint
- minden levél ugyanolyan messze van a gyökértől
- a fa belső csúcsai: a különböző szintű indexek lapjai

B-fa

A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa:

- a fa levelei: a főállomány blokkjai
- a főállomány (a levelek) rendezett a keresési kulcs szerint
- minden levél ugyanolyan messze van a gyökértől
- a fa belső csúcsai: a különböző szintű indexek lapjai
- egy csúcs gyerekei: az indexlapon levő mutatóknak megfelelő eggyel lejjebb levő indexlapok (illetve alul levelek)

B-fa

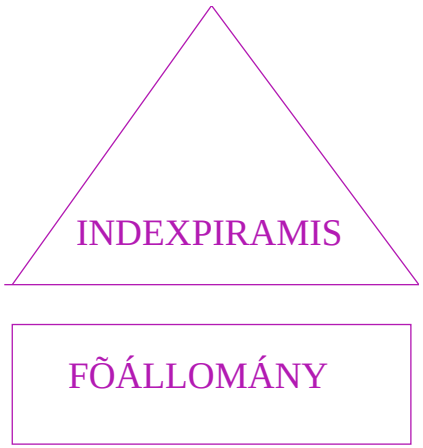
A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa:

- a fa levelei: a főállomány blokkjai
- a főállomány (a levelek) rendezett a keresési kulcs szerint
- minden levél ugyanolyan messze van a gyökértől
- a fa belső csúcsai: a különböző szintű indexek lapjai
- egy csúcs gyerekei: az indexlapon levő mutatóknak megfelelő eggyel lejjebb levő indexlapok (illetve alul levelek)
- m: egy lapra m indexrekord fér rá

B-fa

A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa:

- a fa levelei: a főállomány blokkjai
- a főállomány (a levelek) rendezett a keresési kulcs szerint
- minden levél ugyanolyan messze van a gyökértől
- a fa belső csúcsai: a különböző szintű indexek lapjai
- egy csúcs gyerekei: az indexlapon levő mutatóknak megfelelő eggyel lejjebb levő indexlapok (illetve alul levelek)
- m: egy lapra m indexrekord fér rá
- minden lap legalább félig kitöltött, kivéve esetleg a gyökeret (minden csúcsnak legalább $\frac{m}{2}$ gyereke van, kivéve esetleg a gyökeret)



Műveletek

- **keresés**: a csúcsokban található kulcs-bejegyzések és mutatók mentén; arányos a fa magasságával, ami $O(log_m n)$, ha n blokkja van a főállománynak

Műveletek

- **keresés**: a csúcsokban található kulcs-bejegyzések és mutatók mentén; arányos a fa magasságával, ami $O(log_m n)$, ha n blokkja van a főállománynak
- **beszúrás**: beszúrás után esetleg csúcsvágás(ok), de max. $O(log_m n)$

Műveletek

- **keresés**: a csúcsokban található kulcs-bejegyzések és mutatók mentén; arányos a fa magasságával, ami $O(log_m n)$, ha n blokkja van a főállománynak
- **beszúrás**: beszúrás után esetleg csúcsvágás(ok), de max. $O(log_m n)$
- **törlés**: törlés után esetleg csúcsösszevonás(ok), de max. $O(log_m n)$

Műveletek

- **keresés**: a csúcsokban található kulcs-bejegyzések és mutatók mentén; arányos a fa magasságával, ami $O(log_m n)$, ha n blokkja van a főállománynak
- **beszúrás**: beszúrás után esetleg csúcsvágás(ok), de max. $O(log_m n)$
- **törlés**: törlés után esetleg csúcsösszevonás(ok), de max. $O(log_m n)$

(A műveletek végrehajtásának részleteit az Algoritmuselmélet tárgy taglalja, itt most nem tárgyaljuk.)
Megjegyzések:

1. Ha m nagy $\Longrightarrow$ ritkán kell csúcsvágás/csúcsösszevonás.

Műveletek

- **keresés**: a csúcsokban található kulcs-bejegyzések és mutatók mentén; arányos a fa magasságával, ami $O(log_m n)$, ha n blokkja van a főállománynak
- **beszúrás**: beszúrás után esetleg csúcsvágás(ok), de max. $O(log_m n)$
- **törlés**: törlés után esetleg csúcsösszevonás(ok), de max. $O(log_m n)$

(A műveletek végrehajtásának részleteit az Algoritmuselmélet tárgy taglalja, itt most nem tárgyaljuk.)
Megjegyzések:

1. Ha m nagy $\Longrightarrow$ ritkán kell csúcsvágás/csúcsösszevonás.
2. általában m úgy van választva, hogy a fa magassága max. 4 legyen, ha az első lap a belső memóriában van, akkor elég 3 I/O művelet mindenhez

Sűrű index

Eddig feltettük, hogy a főállomány szabad és (lényegében) rendezett a keresési kulcs szerint. Hogyan érjük ezt el? Hogyan lehet több kulcs szerint is keresni?

Sűrű index

Eddig feltettük, hogy a főállomány szabad és (lényegében) rendezett a keresési kulcs szerint. Hogyan érjük ezt el? Hogyan lehet több kulcs szerint is keresni?
Erre megoldás a sűrű index:

- a főállomány minden rekordjához van egy indexbejegyzés $\Longrightarrow$ ugyanannyi bejegyzés lesz a sűrű indexben, mint ahány rekord van a főállományban, csak persze kisebbek a bejegyzések $\Longrightarrow$ sűrű index = főállomány kicsiben

Sűrű index

Eddig feltettük, hogy a főállomány szabad és (lényegében) rendezett a keresési kulcs szerint. Hogyan érjük ezt el? Hogyan lehet több kulcs szerint is keresni?
Erre megoldás a sűrű index:

- a főállomány minden rekordjához van egy indexbejegyzés $\Longrightarrow$ ugyanannyi bejegyzés lesz a sűrű indexben, mint ahány rekord van a főállományban, csak persze kisebbek a bejegyzések $\Longrightarrow$ sűrű index = főállomány kicsiben
- ez nem önálló állományszervezési technika (ellentétben a ritka index változataival), hanem csak kiegészítés, ami lehetővé teszi, hogy a főállományt szabadnak és rendezettnek tételezhessük fel

Sűrű index

Eddig feltettük, hogy a főállomány szabad és (lényegében) rendezett a keresési kulcs szerint. Hogyan érjük ezt el? Hogyan lehet több kulcs szerint is keresni?
Erre megoldás a sűrű index:

- a főállomány minden rekordjához van egy indexbejegyzés $\Longrightarrow$ ugyanannyi bejegyzés lesz a sűrű indexben, mint ahány rekord van a főállományban, csak persze kisebbek a bejegyzések $\Longrightarrow$ sűrű index = főállomány kicsiben
- ez nem önálló állományszervezési technika (ellentétben a ritka index változataival), hanem csak kiegészítés, ami lehetővé teszi, hogy a főállományt szabadnak és rendezettnek tételezhessük fel

Haszna:

- szabaddá teszi a rekordokat (a főállomány ugyan kötött, de a sűrű index bejegyzései szabadon mozgathatók: építhető rá ritka index)

Sűrű index

Eddig feltettük, hogy a főállomány szabad és (lényegében) rendezett a keresési kulcs szerint. Hogyan érjük ezt el? Hogyan lehet több kulcs szerint is keresni?
Erre megoldás a sűrű index:

- a főállomány minden rekordjához van egy indexbejegyzés $\Longrightarrow$ ugyanannyi bejegyzés lesz a sűrű indexben, mint ahány rekord van a főállományban, csak persze kisebbek a bejegyzések $\Longrightarrow$ sűrű index = főállomány kicsiben
- ez nem önálló állományszervezési technika (ellentétben a ritka index változataival), hanem csak kiegészítés, ami lehetővé teszi, hogy a főállományt szabadnak és rendezettnek tételezhessük fel

Haszna:

- szabaddá teszi a rekordokat (a főállomány ugyan kötött, de a sűrű index bejegyzései szabadon mozgathatók: építhető rá ritka index)
- rendezettnek mutatja a főállományt: a sűrű indexet úgy rendezzük, ahogy akarjuk

Sűrű index

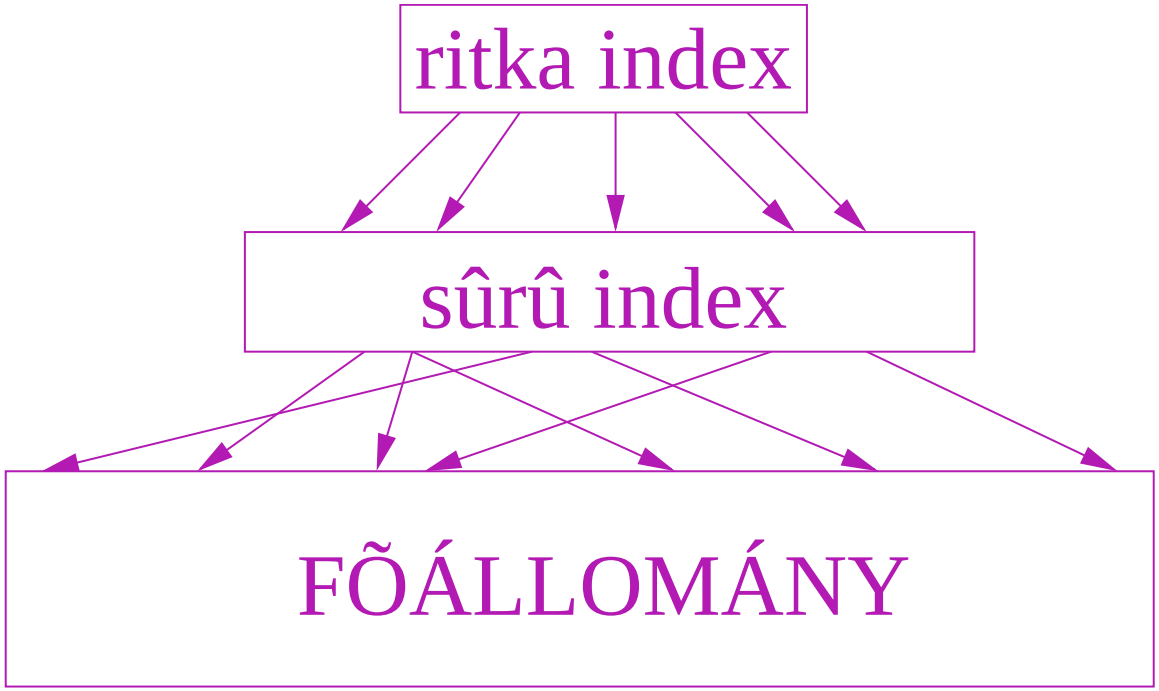
Eddig feltettük, hogy a főállomány szabad és (lényegében) rendezett a keresési kulcs szerint. Hogyan érjük ezt el? Hogyan lehet több kulcs szerint is keresni?
Erre megoldás a sűrű index:

- a főállomány minden rekordjához van egy indexbejegyzés $\Longrightarrow$ ugyanannyi bejegyzés lesz a sűrű indexben, mint ahány rekord van a főállományban, csak persze kisebbek a bejegyzések $\Longrightarrow$ sűrű index = főállomány kicsiben
- ez nem önálló állományszervezési technika (ellentétben a ritka index változataival), hanem csak kiegészítés, ami lehetővé teszi, hogy a főállományt szabadnak és rendezettnek tételezhessük fel

Haszna:

- szabaddá teszi a rekordokat (a főállomány ugyan kötött, de a sűrű index bejegyzései szabadon mozgathatók: építhető rá ritka index)
- rendezettnek mutatja a főállományt: a sűrű indexet úgy rendezzük, ahogy akarjuk
- sokkal kisebb, mint a főállomány, mégis egy az egyben megfelel neki

Tipikus használata:



A sűrű index ráépül a főállományra, erre építjük a valódi állományszervezést. A sűrű index miatt a főállomány szabadnak és rendezettnek tűnik.

Műveletek ebben a struktúrában

Úgy dolgozunk, mintha a sűrű index lenne a főállomány, innen már csak egy plusz lapelérés a valódi főállomány.

Műveletek ebben a struktúrában

Úgy dolgozunk, mintha a sűrű index lenne a főállomány, innen már csak egy plusz lapelérés a valódi főállomány.

- **keresés:** keresés sűrűben, onnan egy lapelérés

Műveletek ebben a struktúrában

Úgy dolgozunk, mintha a sűrű index lenne a főállomány, innen már csak egy plusz lapelérés a valódi főállomány.

- **keresés:** keresés sűrűben, onnan egy lapelérés
- **beszúrás:** beszúrás sűrűbe, aztán valahova berakjuk a főállományba

Műveletek ebben a struktúrában

Úgy dolgozunk, mintha a sűrű index lenne a főállomány, innen már csak egy plusz lapelérés a valódi főállomány.

- **keresés:** keresés sűrűben, onnan egy lapelérés
- **beszúrás:** beszúrás sűrűbe, aztán valahova berakjuk a főállományba
- **törlés**: keresés, törlés a főállományból, törlés a sűrűből is

Műveletek ebben a struktúrában

Úgy dolgozunk, mintha a sűrű index lenne a főállomány, innen már csak egy plusz lapelérés a valódi főállomány.

- **keresés:** keresés sűrűben, onnan egy lapelérés
- **beszúrás:** beszúrás sűrűbe, aztán valahova berakjuk a főállományba
- **törlés**: keresés, törlés a főállományból, törlés a sűrűből is

Hátrány

- plusz egy lapelérés kell a sűrű miatt

Műveletek ebben a struktúrában

Úgy dolgozunk, mintha a sűrű index lenne a főállomány, innen már csak egy plusz lapelérés a valódi főállomány.

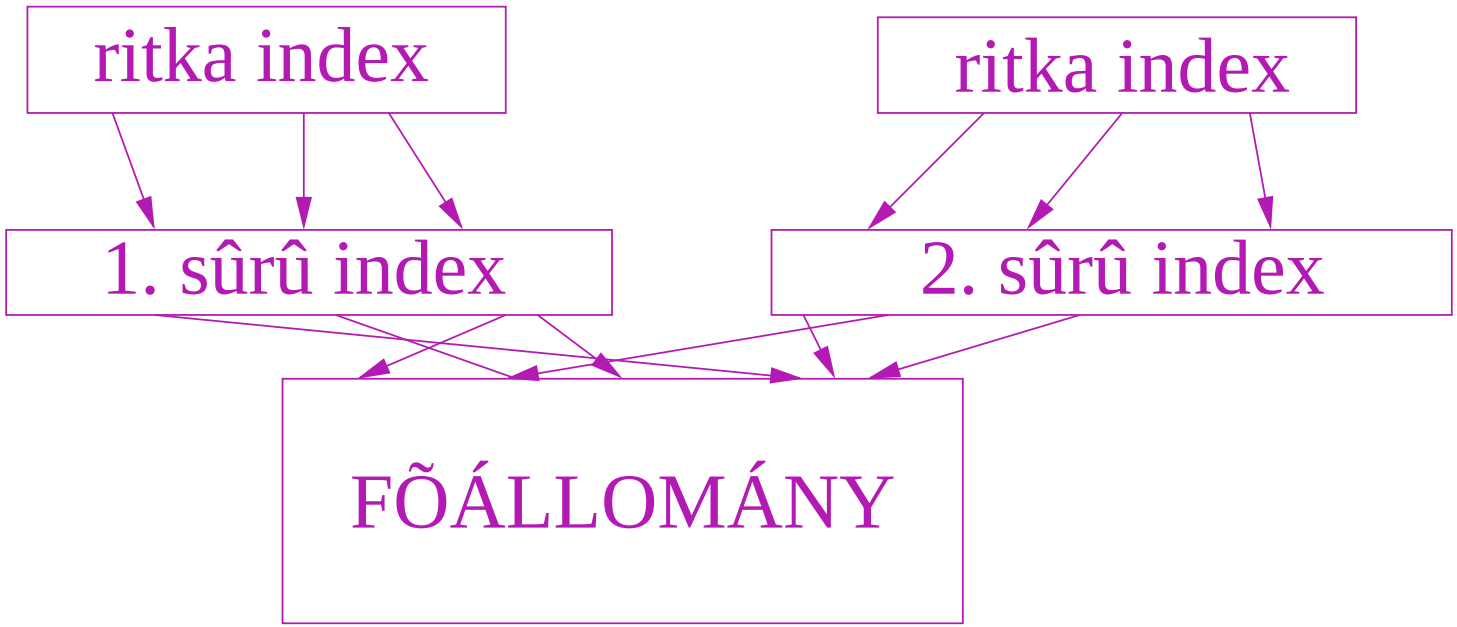
- **keresés:** keresés sűrűben, onnan egy lapelérés
- **beszúrás:** beszúrás sűrűbe, aztán valahova berakjuk a főállományba
- **törlés**: keresés, törlés a főállományból, törlés a sűrűből is

Hátrány

- plusz egy lapelérés kell a sűrű miatt
- karban kell tartani a sűrű indexet is mindig, amikor a főállomány változik

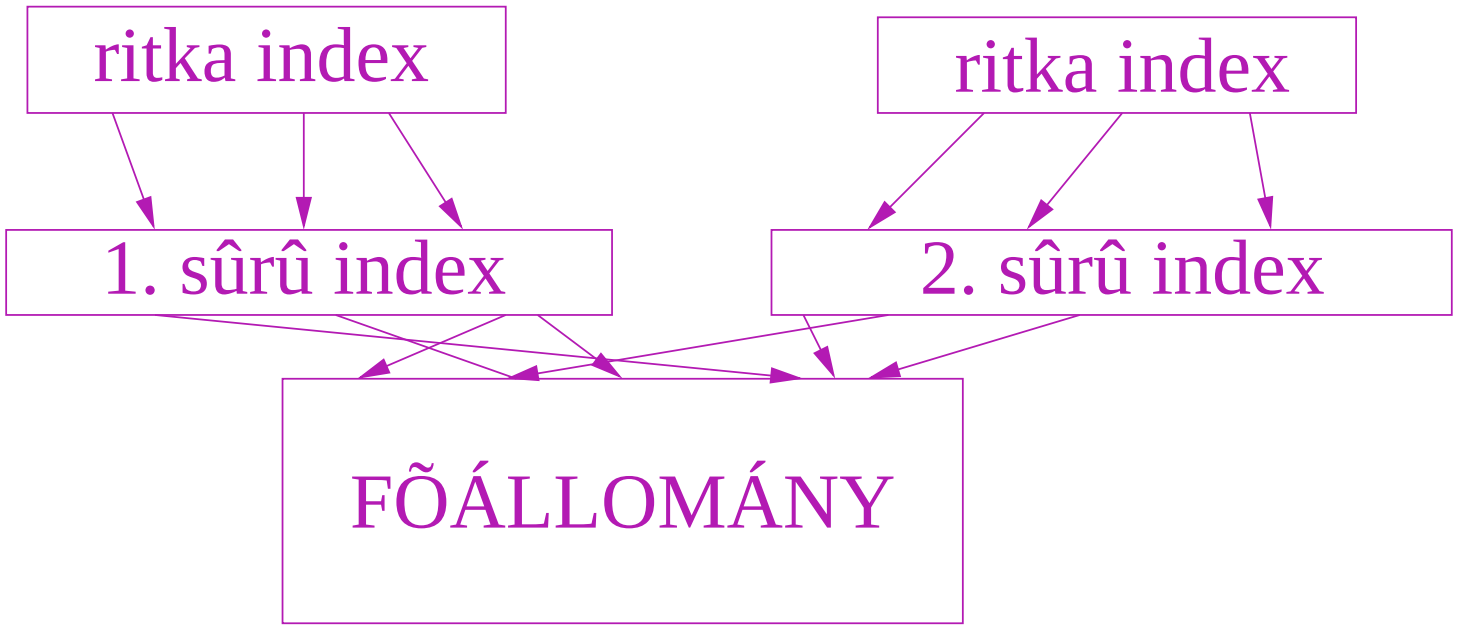
Nagy előnye a sűrű indexnek

Lehetővé teszi, hogy egy főállományra több különböző kulcs szerint is legyen index:



Nagy előnye a sűrű indexnek

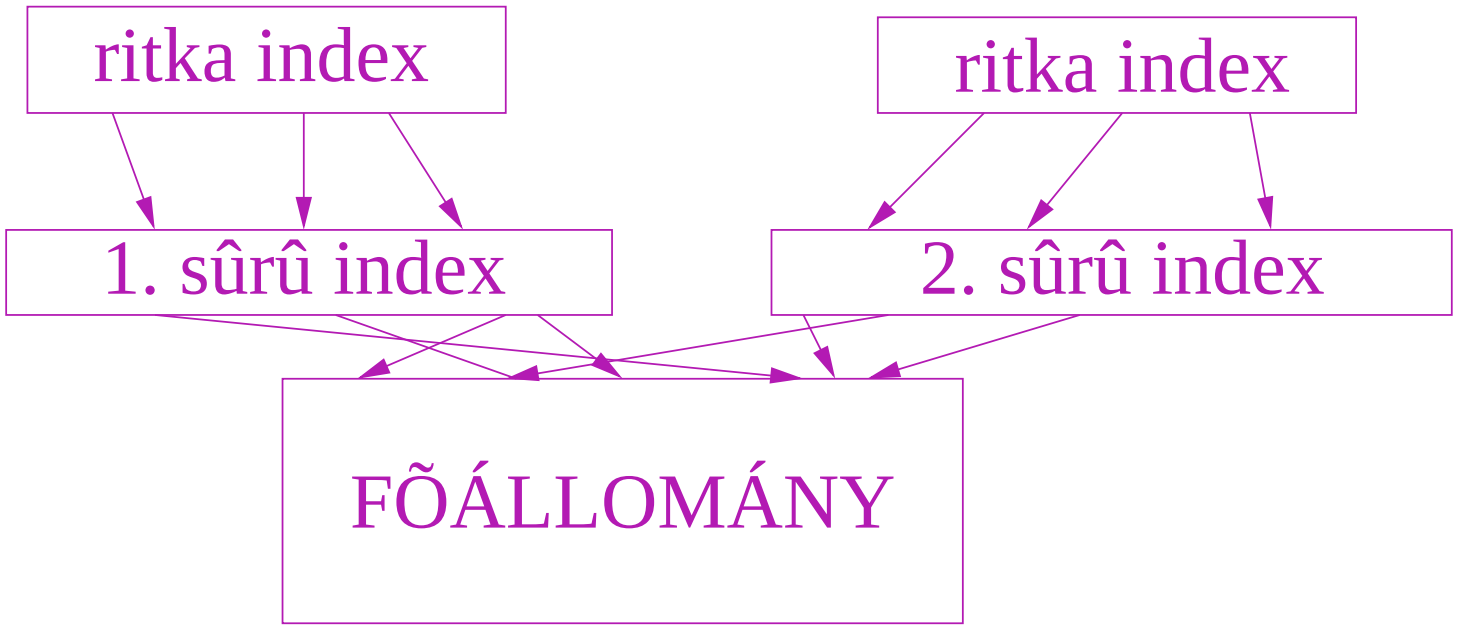
Lehetővé teszi, hogy egy főállományra több különböző kulcs szerint is legyen index:



Itt minden sűrű index rendezett a megfelelő kulcs szerint és persze ha változik a főállomány, akkor mindegyik sűrűt is változtatni kell.

Nagy előnye a sűrű indexnek

Lehetővé teszi, hogy egy főállományra több különböző kulcs szerint is legyen index:



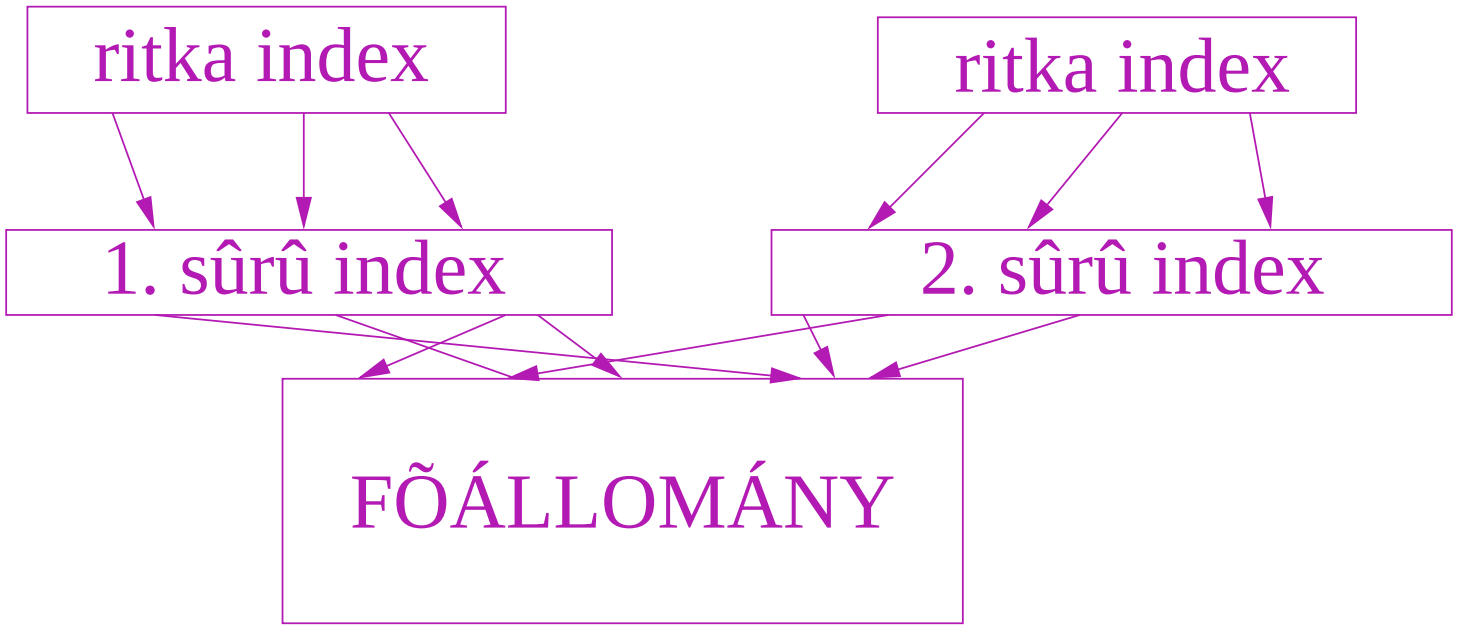
Itt minden sűrű index rendezett a megfelelő kulcs szerint és persze ha változik a főállomány, akkor mindegyik sűrűt is változtatni kell.

Példa:

A Személy(név, telefonszám, személyiszám, . . .) sémában a személyiszám az elsődleges kulcs, ezért a rendszer eszerint rendezetten tárolja az adatokat és erre biztosan létre is hoz valami keresési struktúrát.

Nagy előnye a sűrű indexnek

Lehetővé teszi, hogy egy főállományra több különböző kulcs szerint is legyen index:



Itt minden sűrű index rendezett a megfelelő kulcs szerint és persze ha változik a főállomány, akkor mindegyik sűrűt is változtatni kell.

Példa:

A Személy(név, telefonszám, személyiszám, . . .) sémában a személyiszám az elsődleges kulcs, ezért a rendszer eszerint rendezetten tárolja az adatokat és erre biztosan létre is hoz valami keresési struktúrát.

De ha mi szeretnénk a név-re is: kell egy sűrű index: invertált állomány.

Különböző technikák összevetése

- **hash**: konstans (gyakran 1) lapelérés átlagosan, de legrosszabb esetben lassú

Különböző technikák összevetése

- **hash**: konstans (gyakran 1) lapelérés átlagosan, de legrosszabb esetben lassú
- **ritka index**: korlátos viselkedés legrosszabb esetben is, dinamikus bővülés támogatása, rendezettség figyelembe vétele; B-fa esetén a gyakorlatban konstans lapelérés

Különböző technikák összevetése

- **hash**: konstans (gyakran 1) lapelérés átlagosan, de legrosszabb esetben lassú
- **ritka index**: korlátos viselkedés legrosszabb esetben is, dinamikus bővülés támogatása, rendezettség figyelembe vétele; B-fa esetén a gyakorlatban konstans lapelérés
- **sűrű index**: önmagában nem jó, csak kiegészítésül szolgál

Számolási példa

Egy állományt sűrű index, majd erre épített 1-szintes ritka index segítségével szeretnénk tárolni. Adjon értelmes alsó becslést a szükséges lapok számára az alábbi feltételek mellett:

- az állomány $3 \cdot 10^6$ rekordból áll,
- egy rekord hossza 300 Byte,
- egy lap mérete 1000 Byte,
- a kulcshossz 45 Byte,
- egy mutató hossza 5 Byte.

Számolási példa

Egy állományt sűrű index, majd erre épített 1-szintes ritka index segítségével szeretnénk tárolni. Adjon értelmes alsó becslést a szükséges lapok számára az alábbi feltételek mellett:

- az állomány $3 \cdot 10^6$ rekordból áll,
- egy rekord hossza 300 Byte,
- egy lap mérete 1000 Byte,
- a kulcshossz 45 Byte,
- egy mutató hossza 5 Byte.

Megoldás:
A főállományban $3 \cdot 10^6$ rekord van, mivel rekordok nem lóghatnak át laphatáron, ezért ehhez kell 10^6 lap.

Számolási példa

Egy állományt sűrű index, majd erre épített 1-szintes ritka index segítségével szeretnénk tárolni. Adjon értelmes alsó becslést a szükséges lapok számára az alábbi feltételek mellett:

- az állomány $3 \cdot 10^6$ rekordból áll,
- egy rekord hossza 300 Byte,
- egy lap mérete 1000 Byte,
- a kulcshossz 45 Byte,
- egy mutató hossza 5 Byte.

Megoldás:
A főállományban $3 \cdot 10^6$ rekord van, mivel rekordok nem lóghatnak át laphatáron, ezért ehhez kell 10^6 lap.
A sűrű indexben annyi bejegyzés lesz, mint ahány rekord van a főállományban, azaz $3 \cdot 10^6$.

Számolási példa

Egy állományt sűrű index, majd erre épített 1-szintes ritka index segítségével szeretnénk tárolni. Adjon értelmes alsó becslést a szükséges lapok számára az alábbi feltételek mellett:

- az állomány $3 \cdot 10^6$ rekordból áll,
- egy rekord hossza 300 Byte,
- egy lap mérete 1000 Byte,
- a kulcshossz 45 Byte,
- egy mutató hossza 5 Byte.

Megoldás:
A főállományban $3 \cdot 10^6$ rekord van, mivel rekordok nem lóghatnak át laphatáron, ezért ehhez kell 10^6 lap.
A sűrű indexben annyi bejegyzés lesz, mint ahány rekord van a főállományban, azaz $3 \cdot 10^6$.
Egy lapra pontosan 20 bejegyzés fér: ez $1,5 \cdot 10^5$ lap.

Számolási példa

Egy állományt sűrű index, majd erre épített 1-szintes ritka index segítségével szeretnénk tárolni. Adjon értelmes alsó becslést a szükséges lapok számára az alábbi feltételek mellett:

- az állomány $3 \cdot 10^6$ rekordból áll,
- egy rekord hossza 300 Byte,
- egy lap mérete 1000 Byte,
- a kulcshossz 45 Byte,
- egy mutató hossza 5 Byte.

Megoldás:
A főállományban $3 \cdot 10^6$ rekord van, mivel rekordok nem lóghatnak át laphatáron, ezért ehhez kell 10^6 lap.
A sűrű indexben annyi bejegyzés lesz, mint ahány rekord van a főállományban, azaz $3 \cdot 10^6$.
Egy lapra pontosan 20 bejegyzés fér: ez $1,5 \cdot 10^5$ lap.
Ez azt is jelenti, hogy a ritka indexben lesz legalább $1,5 \cdot 10^5$ bejegyzés, ehhez kell még $7,5 \cdot 10^3$ lap.

Számolási példa

Egy állományt sűrű index, majd erre épített 1-szintes ritka index segítségével szeretnénk tárolni. Adjon értelmes alsó becslést a szükséges lapok számára az alábbi feltételek mellett:

- az állomány $3 \cdot 10^6$ rekordból áll,
- egy rekord hossza 300 Byte,
- egy lap mérete 1000 Byte,
- a kulcshossz 45 Byte,
- egy mutató hossza 5 Byte.

Megoldás:
A főállományban $3 \cdot 10^6$ rekord van, mivel rekordok nem lóghatnak át laphatáron, ezért ehhez kell 10^6 lap.
A sűrű indexben annyi bejegyzés lesz, mint ahány rekord van a főállományban, azaz $3 \cdot 10^6$.
Egy lapra pontosan 20 bejegyzés fér: ez $1,5 \cdot 10^5$ lap.
Ez azt is jelenti, hogy a ritka indexben lesz legalább $1,5 \cdot 10^5$ bejegyzés, ehhez kell még $7,5 \cdot 10^3$ lap.
Ez összesen 1 157 500 lap.

Adatbázisok elmélete

Tranzakciókezelés

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

21. előadás

Tranzakciókezelés

Eddig hallgatólagosan feltettük, hogy

- egy felhasználó van csak

Tranzakciókezelés

Eddig hallgatólagosan feltettük, hogy

- egy felhasználó van csak
- a lekérdezések/módosítások hiba nélkül lefutnak

Tranzakciókezelés

Eddig hallgatólagosan feltettük, hogy

- egy felhasználó van csak
- a lekérdezések/módosítások hiba nélkül lefutnak

A valóságban ez nincs így, két nagyobb gond is lehet, aminek kezelése a tranzakciókezelő dolga:

- **Többfelhasználós működés**: egyidejű hozzáférést kell biztosítani több felhasználónak, de úgy, hogy az adatbázis konzisztens maradjon (pl. banki rendszerek, helyfoglalás)

Tranzakciókezelés

Eddig hallgatólagosan feltettük, hogy

- egy felhasználó van csak
- a lekérdezések/módosítások hiba nélkül lefutnak

A valóságban ez nincs így, két nagyobb gond is lehet, aminek kezelése a tranzakciókezelő dolga:

- **Többfelhasználós működés**: egyidejű hozzáférést kell biztosítani több felhasználónak, de úgy, hogy az adatbázis konzisztens maradjon (pl. banki rendszerek, helyfoglalás)
- **Rendszerhibák utáni helyreállítás**: ha a külső tár megmarad, de a belső sérül (vagy egyszerűen csak nem fut le valami) és emiatt az adatbázis inkonzisztens állapotba kerül, akkor újra konzisztens állapotba kell hozni (vagy visszacsinálni valamit, vagy befejezni valamit)

Tranzakciókezelés

Eddig hallgatólagosan feltettük, hogy

- egy felhasználó van csak
- a lekérdezések/módosítások hiba nélkül lefutnak

A valóságban ez nincs így, két nagyobb gond is lehet, aminek kezelése a tranzakciókezelő dolga:

- **Többfelhasználós működés**: egyidejű hozzáférést kell biztosítani több felhasználónak, de úgy, hogy az adatbázis konzisztens maradjon (pl. banki rendszerek, helyfoglalás)
- **Rendszerhibák utáni helyreállítás**: ha a külső tár megmarad, de a belső sérül (vagy egyszerűen csak nem fut le valami) és emiatt az adatbázis inkonzisztens állapotba kerül, akkor újra konzisztens állapotba kell hozni (vagy visszacsinálni valamit, vagy befejezni valamit)

Ez két (néha egymással is ellentétes) kívánság, de az alapeszköz ugyanaz lesz: a **tranzakció**.

Többfelhasználós működés

A lekérdezésfeldolgozó a magas szintű utasításokból álló lekérdezéseket/módosításokat elemi utasításokra bontja, (pl: olvass ki valahonnan valamit, írj be valahova valamit, számolj valamit). Egy felhasználó egy lekérdezése/módosítása ilyen elemi utasítások sorozatává alakul.

Többfelhasználós működés

A lekérdezésfeldolgozó a magas szintű utasításokból álló lekérdezéseket/módosításokat elemi utasításokra bontja, (pl: olvass ki valahonnan valamit, írj be valahova valamit, számolj valamit). Egy felhasználó egy lekérdezése/módosítása ilyen elemi utasítások sorozatává alakul.

1. felhasználó: $u_1, u_2, \ldots, u_{10}$
2. felhasználó: $v_1, v_2, \ldots, v_{103}$

Többfelhasználós működés

A lekérdezésfeldolgozó a magas szintű utasításokból álló lekérdezéseket/módosításokat elemi utasításokra bontja, (pl: olvass ki valahonnan valamit, írj be valahova valamit, számolj valamit). Egy felhasználó egy lekérdezése/módosítása ilyen elemi utasítások sorozatává alakul.

1. felhasználó: $u_1, u_2, \ldots, u_{10}$
2. felhasználó: $v_1, v_2, \ldots, v_{103}$

De ez a két utasítássorozat nem elkülönülve jön, hanem összefésülődnek:

$$u_1, v_1, v_2, u_2, u_3, v_3, \ldots, v_{103}, u_{10}$$

Többfelhasználós működés

A lekérdezésfeldolgozó a magas szintű utasításokból álló lekérdezéseket/módosításokat elemi utasításokra bontja, (pl: olvass ki valahonnan valamit, írj be valahova valamit, számolj valamit). Egy felhasználó egy lekérdezése/módosítása ilyen elemi utasítások sorozatává alakul.

1. felhasználó: $u_1, u_2, \ldots, u_{10}$
2. felhasználó: $v_1, v_2, \ldots, v_{103}$

De ez a két utasítássorozat nem elkülönülve jön, hanem összefésülődnek:

$$u_1, v_1, v_2, u_2, u_3, v_3, \ldots, v_{103}, u_{10}$$

A saját sorrend megmarad mindkettőn belül, de amúgy össze vannak keveredve, így lesz lehetséges a több felhasználó egyidejű kiszolgálása.

Többfelhasználós működés

A lekérdezésfeldolgozó a magas szintű utasításokból álló lekérdezéseket/módosításokat elemi utasításokra bontja, (pl: olvass ki valahonnan valamit, írj be valahova valamit, számolj valamit). Egy felhasználó egy lekérdezése/módosítása ilyen elemi utasítások sorozatává alakul.

1. felhasználó: $u_1, u_2, \ldots, u_{10}$
2. felhasználó: $v_1, v_2, \ldots, v_{103}$

De ez a két utasítássorozat nem elkülönülve jön, hanem összefésülődnek:

$$u_1, v_1, v_2, u_2, u_3, v_3, \ldots, v_{103}, u_{10}$$

A saját sorrend megmarad mindkettőn belül, de amúgy össze vannak keveredve, így lesz lehetséges a több felhasználó egyidejű kiszolgálása. Ebből viszont baj származhat, mert olyan állapot is kialakulhat, ami nem jött volna létre, ha egymás után futnak le a tranzakciók.

Példa

1. felhasználó: READ A, $A++$, WRITE A
2. felhasználó: READ A, $A++$, WRITE A

Példa

1. felhasználó: READ A, $A++$, WRITE A
2. felhasználó: READ A, $A++$, WRITE A

Ha ezek úgy fésülődnek össze, hogy

$$(\text{READ } A)_1, (\text{READ } A)_2, (A++)_1, (A++)_2, (\text{WRITE } A)_1, (\text{WRITE } A)_2$$

Példa

1. felhasználó: READ A, $A++$, WRITE A
2. felhasználó: READ A, $A++$, WRITE A

Ha ezek úgy fésülődnek össze, hogy

$$(\text{READ } A)_1, (\text{READ } A)_2, (A++)_1, (A++)_2, (\text{WRITE } A)_1, (\text{WRITE } A)_2$$

akkor a végén csak eggyel nő A értéke, holott kettővel kellett volna.

Rendszerhibák

Ha rendszerhiba van (a belső memória meghibásodik) vagy csak ABORT van (a tranzakciókezelő ütemező része kilő egy alkalmazást futás közben), akkor emiatt félbemaradhat valami, aminek nem lenne szabad.

Rendszerhibák

Ha rendszerhiba van (a belső memória meghibásodik) vagy csak ABORT van (a tranzakciókezelő ütemező része kilő egy alkalmazást futás közben), akkor emiatt félbemaradhat valami, aminek nem lenne szabad.

Példa: átutalunk egyik helyről a másik helyre pénzt:

$$A := A - 50 \quad B := B + 50$$

Rendszerhibák

Ha rendszerhiba van (a belső memória meghibásodik) vagy csak ABORT van (a tranzakciókezelő ütemező része kilő egy alkalmazást futás közben), akkor emiatt félbemaradhat valami, aminek nem lenne szabad.

Példa: átutalunk egyik helyről a másik helyre pénzt:

$$A := A - 50 \quad B := B + 50$$

Ha az a közepén meghal: hibás állapot jön létre.

Tranzakció

Alapfogalom mindkét problémakör megoldásában a **tranzakció**: egy felhasználóhoz tartozó elemi utasítások olyan sorozata, melynek fő jellemzője az **atomiság** (**A**tomicity): vagy az összes utasításnak végre kell hajtódnia vagy egynek sem szabad. Ez lesz az egyik dolog, amit mindenáron el akarunk majd érni.

Tranzakció

Alapfogalom mindkét problémakör megoldásában a **tranzakció**: egy felhasználóhoz tartozó elemi utasítások olyan sorozata, melynek fő jellemzője az **atomiság** (**A**tomicity): vagy az összes utasításnak végre kell hajtódnia vagy egynek sem szabad. Ez lesz az egyik dolog, amit mindenáron el akarunk majd érni.

További elvárások:

- **konzisztencia**, **C**onsistency: az adatbázis konzisztens állapotok között mozog, (hogy mit jelent a konzisztens, az a valóságtól függ, pl. banki összegek stimmelése), nem konzisztens állapot csak ideiglenesen állhat fenn (a rendszerhibák utáni helyreállításnál lesz ez fontos)

Tranzakció

Alapfogalom mindkét problémakör megoldásában a **tranzakció**: egy felhasználóhoz tartozó elemi utasítások olyan sorozata, melynek fő jellemzője az **atomiság** (**A**tomicity): vagy az összes utasításnak végre kell hajtódnia vagy egynek sem szabad. Ez lesz az egyik dolog, amit mindenáron el akarunk majd érni.

További elvárások:

- **konzisztencia**, **C**onsistency: az adatbázis konzisztens állapotok között mozog, (hogy mit jelent a konzisztens, az a valóságtól függ, pl. banki összegek stimmelése), nem konzisztens állapot csak ideiglenesen állhat fenn (a rendszerhibák utáni helyreállításnál lesz ez fontos)
- **elkülönítés**, **I**solation: több tranzakció egyidejű futása után úgy kell kinéznie az adatbázisnak, mintha a tranzakciók nem lettek volna összefésülve (az ütemező dolga lesz ennek biztosítása)

Tranzakció

Alapfogalom mindkét problémakör megoldásában a **tranzakció**: egy felhasználóhoz tartozó elemi utasítások olyan sorozata, melynek fő jellemzője az **atomiság** (**A**tomicity): vagy az összes utasításnak végre kell hajtódnia vagy egynek sem szabad. Ez lesz az egyik dolog, amit mindenáron el akarunk majd érni.

További elvárások:

- **konzisztencia**, **C**onsistency: az adatbázis konzisztens állapotok között mozog, (hogy mit jelent a konzisztens, az a valóságtól függ, pl. banki összegek stimmelése), nem konzisztens állapot csak ideiglenesen állhat fenn (a rendszerhibák utáni helyreállításnál lesz ez fontos)
- **elkülönítés**, **I**solation: több tranzakció egyidejű futása után úgy kell kinéznie az adatbázisnak, mintha a tranzakciók nem lettek volna összefésülve (az ütemező dolga lesz ennek biztosítása)
- **tartósság**, **D**urability: a befejezett tranzakciók hatása nem veszhet el

Többfelhasználós működés, alapfogalmak

Cél: párhuzamos hozzáférés biztosítása, de úgy, hogy a konzisztencia megmaradjon

Többfelhasználós működés, alapfogalmak

Cél: párhuzamos hozzáférés biztosítása, de úgy, hogy a konzisztencia megmaradjon
Feltételezzük, hogy ha a tranzakciók egymás után, elkülönítve futnak, akkor konzisztens állapotból konzisztens állapotba jut a rendszer. Csak azokat az összefésülődéseit akarjuk megengedni a tranzakcióknak, amelyeknek a hatása ekvivalens valamelyik izolálttal.

Többfelhasználós működés, alapfogalmak

Cél: párhuzamos hozzáférés biztosítása, de úgy, hogy a konzisztencia megmaradjon
Feltételezzük, hogy ha a tranzakciók egymás után, elkülönítve futnak, akkor konzisztens állapotból konzisztens állapotba jut a rendszer. Csak azokat az összefésülődéseit akarjuk megengedni a tranzakcióknak, amelyeknek a hatása ekvivalens valamelyik izolálttal.

ütemezés: egy vagy több tranzakció műveleteinek valamilyen sorozata (fontos, hogy a tranzakciókon belüli sorrend megmarad)

Többfelhasználós működés, alapfogalmak

Cél: párhuzamos hozzáférés biztosítása, de úgy, hogy a konzisztencia megmaradjon
Feltételezzük, hogy ha a tranzakciók egymás után, elkülönítve futnak, akkor konzisztens állapotból konzisztens állapotba jut a rendszer. Csak azokat az összefésülődéseit akarjuk megengedni a tranzakcióknak, amelyeknek a hatása ekvivalens valamelyik izolálttal.

ütemezés: egy vagy több tranzakció műveleteinek valamilyen sorozata (fontos, hogy a tranzakciókon belüli sorrend megmarad)

soros ütemezés:olyan ütemezés, amikor a különböző tranzakciók utasításai nem keverednek, először lefut az egyik összes utasítása, aztán a másiké, aztán a harmadiké, . . .

Többfelhasználós működés, alapfogalmak

Cél: párhuzamos hozzáférés biztosítása, de úgy, hogy a konzisztencia megmaradjon
Feltételezzük, hogy ha a tranzakciók egymás után, elkülönítve futnak, akkor konzisztens állapotból konzisztens állapotba jut a rendszer. Csak azokat az összefésülődéseit akarjuk megengedni a tranzakcióknak, amelyeknek a hatása ekvivalens valamelyik izolálttal.

ütemezés: egy vagy több tranzakció műveleteinek valamilyen sorozata (fontos, hogy a tranzakciókon belüli sorrend megmarad)

soros ütemezés:olyan ütemezés, amikor a különböző tranzakciók utasításai nem keverednek, először lefut az egyik összes utasítása, aztán a másiké, aztán a harmadiké, . . .

sorosítható ütemezés: olyan ütemezés, amelynek hatása azonos a résztvevő tranzakciók **valamely** soros ütemezésének hatásával (azaz a végén minden érintett adatelem pont úgy néz ki, mint a soros ütemezés után)

Adatbázisok elmélete

Sorosíthatóság

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

22. előadás

Sorosíthatóság

Megjegyzés: Az mindegy, hogy melyik soros ütemezéssel lesz ekvivalens a sorosítható ütemezés. Mivel a soros ütemezésekről feltettük, hogy jók, ezért ha valamelyikkel ekvivalens, az már elég.

Sorosíthatóság

Megjegyzés: Az mindegy, hogy melyik soros ütemezéssel lesz ekvivalens a sorosítható ütemezés. Mivel a soros ütemezésekről feltettük, hogy jók, ezért ha valamelyikkel ekvivalens, az már elég.

Cél: olyan sorrend (összefésülődés) kikényszerítése, ami sorosítható ütemezés

Sorosíthatóság

Megjegyzés: Az mindegy, hogy melyik soros ütemezéssel lesz ekvivalens a sorosítható ütemezés. Mivel a soros ütemezésekről feltettük, hogy jók, ezért ha valamelyikkel ekvivalens, az már elég.

Cél: olyan sorrend (összefésülődés) kikényszerítése, ami sorosítható ütemezés

Módszer: az ütemező (az adatbáziskezelő része) felelős azért, hogy csak ilyen sorrendek legyenek. Figyeli a tranzakciók műveleteit és késleltet/ABORT-ál tranzakciókat. (Nemsokára részletesebben is nézzük.)

Alapfeltevések

- feltesszük, hogy a tranzakciók elemi műveletei: adat olvasása (READ A), számolás az adattal (pl. $A++$), adat írása (WRITE A)

Alapfeltevések

- feltesszük, hogy a tranzakciók elemi műveletei: adat olvasása (READ A), számolás az adattal (pl. $A++$), adat írása (WRITE A)
- a fenti elemi utasításokat tartalmazó műveletsort a lekérdezésfeldolgozó állítja elő, elemzi a magas szintű lekérdezést/módosítást és azt ilyen elemi utasításokból álló sorozattá alakítja

Alapfeltevések

- feltesszük, hogy a tranzakciók elemi műveletei: adat olvasása (READ A), számolás az adattal (pl. $A++$), adat írása (WRITE A)
- a fenti elemi utasításokat tartalmazó műveletsort a lekérdezésfeldolgozó állítja elő, elemzi a magas szintű lekérdezést/módosítást és azt ilyen elemi utasításokból álló sorozattá alakítja
- természetesen megengedett, hogy több helyről olvassunk, mielőtt számolunk, megengedett, hogy több adatból számoljunk ki valamit, illetve, hogy úgy írjunk, hogy nem is olvastuk ki azt az adatot

Alapfeltevések

- feltesszük, hogy a tranzakciók elemi műveletei: adat olvasása (READ A), számolás az adattal (pl. $A++$), adat írása (WRITE A)
- a fenti elemi utasításokat tartalmazó műveletsort a lekérdezésfeldolgozó állítja elő, elemzi a magas szintű lekérdezést/módosítást és azt ilyen elemi utasításokból álló sorozattá alakítja
- természetesen megengedett, hogy több helyről olvassunk, mielőtt számolunk, megengedett, hogy több adatból számoljunk ki valamit, illetve, hogy úgy írjunk, hogy nem is olvastuk ki azt az adatot
- ha egy A adatelemet ki kell olvasni, akkor ha már a belső memóriában (pufferban) van, akkor onnan olvasunk, különben a tárkezelővel még be kell előbb hozni a háttértárról

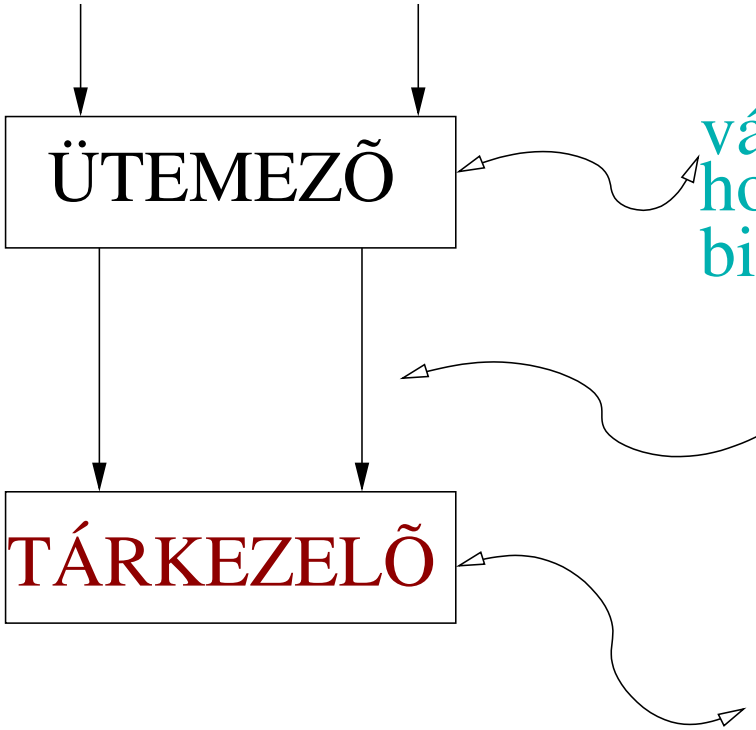
Alapfeltevések

- feltesszük, hogy a tranzakciók elemi műveletei: adat olvasása (READ A), számolás az adattal (pl. $A++$), adat írása (WRITE A)
- a fenti elemi utasításokat tartalmazó műveletsort a lekérdezésfeldolgozó állítja elő, elemzi a magas szintű lekérdezést/módosítást és azt ilyen elemi utasításokból álló sorozattá alakítja
- természetesen megengedett, hogy több helyről olvassunk, mielőtt számolunk, megengedett, hogy több adatból számoljunk ki valamit, illetve, hogy úgy írjunk, hogy nem is olvastuk ki azt az adatot
- ha egy A adatelemet ki kell olvasni, akkor ha már a belső memóriában (pufferban) van, akkor onnan olvasunk, különben a tárkezelővel még be kell előbb hozni a háttértárról
- ha írjuk az A adatelemet, akkor alapértelmezés szerint a pufferbe írjuk, kivéve speciális eseteket, amikor elő lesz írva, hogy valami azonnal kerüljön ki biztonságos háttértárra

A tárkezelővel való együttműködés

Az előbbiek miatt az ütemező és a tárkezelő szorosan együttműködnek:

kérések a tranzakcióktól, írásra, olvasásra



várakoztat, abortot rendel el, hogy a sorosíthatóságot biztosítsa

a tárkezelõ felé továbbküldi az írási és olvasási kéréseket, esetenként pontos elõírással, hogy mit kell a háttértárra írni

megcsinálja, amit az ütemezõ kér

Az ütemező eszközei a sorosíthatóság elérésére

Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorosítható ütemezéseket:

- zárak (ezen belül is még: protokoll elemek, pl. 2PL)

Az ütemező eszközei a sorosíthatóság elérésére

Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorosítható ütemezéseket:

- zárak (ezen belül is még: protokoll elemek, pl. 2PL)
- időbélyegek (time stamp)

Az ütemező eszközei a sorosíthatóság elérésére

Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorosítható ütemezéseket:

- zárak (ezen belül is még: protokoll elemek, pl. 2PL)
- időbélyegek (time stamp)
- érvényesítés

Az ütemező eszközei a sorosíthatóság elérésére

Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorosítható ütemezéseket:

- zárak (ezen belül is még: protokoll elemek, pl. 2PL)
- időbélyegek (time stamp)
- érvényesítés

Fő elv lesz: inkább legyen szigorúbb és ne hagyjon lefutni egy olyat, ami sorosítható, mint hogy fusson egy olyan, aki nem az.

Az ütemező eszközei a sorosíthatóság elérésére

Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorosítható ütemezéseket:

- zárak (ezen belül is még: protokoll elemek, pl. 2PL)
- időbélyegek (time stamp)
- érvényesítés

Fő elv lesz: inkább legyen szigorúbb és ne hagyjon lefutni egy olyat, ami sorosítható, mint hogy fusson egy olyan, aki nem az.
Mindegyik technikára igaz lesz, hogy biztosra megy, azaz olyanokat is ki fog lőni, amik sorosíthatók lennének.

Példa

T_1	T_2	A	B
		x	y

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		
	$s := 2 \cdot s$		

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot (y + 100)$

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot (y + 100)$

A táblázat baloldali részén azt jelezzük, hogy milyen műveleteket végeznek a tranzakciók, a jobboldalon pedig az látszik, hogy eközben mi történik az A és B adategységekkel. Ezek kezdeti értékei x és y.

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot (y + 100)$

A táblázat baloldali részén azt jelezzük, hogy milyen műveleteket végeznek a tranzakciók, a jobboldalon pedig az látszik, hogy eközben mi történik az A és B adategységekkel. Ezek kezdeti értékei x és y.

Read(A,t)= olvassuk be A értékét a t változóba

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot (y + 100)$

A táblázat baloldali részén azt jelezzük, hogy milyen műveleteket végeznek a tranzakciók, a jobboldalon pedig az látszik, hogy eközben mi történik az A és B adategységekkel. Ezek kezdeti értékei x és y.

Read(A,t)= olvassuk be A értékét a t változóba
Write(A,t)= írjuk ki a t változó értékét A-ba

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot (y + 100)$

A táblázat baloldali részén azt jelezzük, hogy milyen műveleteket végeznek a tranzakciók, a jobboldalon pedig az látszik, hogy eközben mi történik az A és B adategységekkel. Ezek kezdeti értékei x és y.

Read(A,t)= olvassuk be A értékét a t változóba
Write(A,t)= írjuk ki a t változó értékét A-ba

Látható, hogy ez nem egy soros ütemezés, mert össze vannak fésülődve a két tranzakció utasításai.

Példa

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
Read(B,t)			
$t := t + 100$			
Write(B,t)			$y + 100$
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot (y + 100)$

A táblázat baloldali részén azt jelezzük, hogy milyen műveleteket végeznek a tranzakciók, a jobboldalon pedig az látszik, hogy eközben mi történik az A és B adategységekkel. Ezek kezdeti értékei x és y.

Read(A,t)= olvassuk be A értékét a t változóba
Write(A,t)= írjuk ki a t változó értékét A-ba

Látható, hogy ez nem egy soros ütemezés, mert össze vannak fésülődve a két tranzakció utasításai.
Viszont sorosítható, mert a hatása A-n és B-n is azonos a $T_1\,T_2$ soros ütemezés hatásával, (x, y)-ból $(2 \cdot (x + 100), 2 \cdot (y + 100))$ lesz.

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$
Read(B,t)			

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$
Read(B,t)			
$t := t + 100$			

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$
Read(B,t)			
$t := t + 100$			
Write(B,t)			$2 \cdot y + 100$

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$
Read(B,t)			
$t := t + 100$			
Write(B,t)			$2 \cdot y + 100$

Ez nem egy sorosítható ütemezés, mert se a $T_1 T_2$ soros ütemezés, se a $T_2 T_1$ soros ütemezés hatása nem az, hogy (x, y)-ból $(2 \cdot (x + 100), 2 \cdot y + 100)$ lesz.

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$
Read(B,t)			
$t := t + 100$			
Write(B,t)			$2 \cdot y + 100$

Ez nem egy sorosítható ütemezés, mert se a $T_1 T_2$ soros ütemezés, se a $T_2 T_1$ soros ütemezés hatása nem az, hogy (x, y)-ból $(2 \cdot (x + 100), 2 \cdot y + 100)$ lesz.
A $T_1 T_2$ ütemezés $(2 \cdot (x + 100), 2 \cdot (y + 100))$ eredményt ad,

Példa nem sorosíthatóra

T_1	T_2	A	B
		x	y
Read(A,t)			
$t := t + 100$			
Write(A,t)		$x + 100$	
	Read(A,s)		
	$s := 2 \cdot s$		
	Write(A,s)	$2 \cdot (x + 100)$	
	Read(B,s)		
	$s := 2 \cdot s$		
	Write(B,s)		$2 \cdot y$
Read(B,t)			
$t := t + 100$			
Write(B,t)			$2 \cdot y + 100$

Ez nem egy sorosítható ütemezés, mert se a $T_1 T_2$ soros ütemezés, se a $T_2 T_1$ soros ütemezés hatása nem az, hogy (x, y)-ból $(2 \cdot (x + 100), 2 \cdot y + 100)$ lesz.
A $T_1 T_2$ ütemezés $(2 \cdot (x + 100), 2 \cdot (y + 100))$ eredményt ad,
a $T_2 T_1$ pedig $(2 \cdot x + 100, 2 \cdot y + 100)$-t.

Egyszerűsítések

Ha ismert, hogy mikor és mit akarnak írni és olvasni a tranzakciók és még az is ismert, hogy pontosan mit számolnak, akkor minden esetben el tudjuk dönteni, hogy egy ütemezés sorosítható-e.

Egyszerűsítések

Ha ismert, hogy mikor és mit akarnak írni és olvasni a tranzakciók és még az is ismert, hogy pontosan mit számolnak, akkor minden esetben el tudjuk dönteni, hogy egy ütemezés sorosítható-e.
A gyakorlatban azonban nem vizsgáljuk meg ennyire alaposan a történéseket, (mert pl. nem is tudnánk vagy mert az macerás), hanem az alábbi egyszerűsítésekkel dolgozunk:

Egyszerűsítések

Ha ismert, hogy mikor és mit akarnak írni és olvasni a tranzakciók és még az is ismert, hogy pontosan mit számolnak, akkor minden esetben el tudjuk dönteni, hogy egy ütemezés sorosítható-e.
A gyakorlatban azonban nem vizsgáljuk meg ennyire alaposan a történéseket, (mert pl. nem is tudnánk vagy mert az macerás), hanem az alábbi egyszerűsítésekkel dolgozunk:

- Nem vizsgáljuk meg, hogy mit számolnak a tranzakciók, hanem feltételezzük a legrosszabbat: valami olyat csinálnak a beolvasott adattal, ami teljesen egyedi. Azaz, feltesszük, hogy ha tud olyat csinálni, amitől inkonzisztens lesz a DB (az ütemezés hatása nem lesz azonos valamelyik soroséval), akkor azt teszi. $\Longrightarrow$

Egyszerűsítések

Ha ismert, hogy mikor és mit akarnak írni és olvasni a tranzakciók és még az is ismert, hogy pontosan mit számolnak, akkor minden esetben el tudjuk dönteni, hogy egy ütemezés sorosítható-e.
A gyakorlatban azonban nem vizsgáljuk meg ennyire alaposan a történéseket, (mert pl. nem is tudnánk vagy mert az macerás), hanem az alábbi egyszerűsítésekkel dolgozunk:

- Nem vizsgáljuk meg, hogy mit számolnak a tranzakciók, hanem feltételezzük a legrosszabbat: valami olyat csinálnak a beolvasott adattal, ami teljesen egyedi. Azaz, feltesszük, hogy ha tud olyat csinálni, amitől inkonzisztens lesz a DB (az ütemezés hatása nem lesz azonos valamelyik soroséval), akkor azt teszi. $\Longrightarrow$
- Csak az írásokat és olvasásokat tartjuk nyilván, ezek alapján döntünk arról, hogy egy ütemezést sorosíthatónak tekintünk-e. Ha csak egyetlen olyan lehetséges számolás is van, amivel az írásokból és olvasásokból álló ütemezés nem sorosítható, akkor nem tekintjük sorosíthatónak.

Egyszerűsítések

Ha ismert, hogy mikor és mit akarnak írni és olvasni a tranzakciók és még az is ismert, hogy pontosan mit számolnak, akkor minden esetben el tudjuk dönteni, hogy egy ütemezés sorosítható-e.
A gyakorlatban azonban nem vizsgáljuk meg ennyire alaposan a történéseket, (mert pl. nem is tudnánk vagy mert az macerás), hanem az alábbi egyszerűsítésekkel dolgozunk:

- Nem vizsgáljuk meg, hogy mit számolnak a tranzakciók, hanem feltételezzük a legrosszabbat: valami olyat csinálnak a beolvasott adattal, ami teljesen egyedi. Azaz, feltesszük, hogy ha tud olyat csinálni, amitől inkonzisztens lesz a DB (az ütemezés hatása nem lesz azonos valamelyik soroséval), akkor azt teszi. $\Longrightarrow$
- Csak az írásokat és olvasásokat tartjuk nyilván, ezek alapján döntünk arról, hogy egy ütemezést sorosíthatónak tekintünk-e. Ha csak egyetlen olyan lehetséges számolás is van, amivel az írásokból és olvasásokból álló ütemezés nem sorosítható, akkor nem tekintjük sorosíthatónak.
- Ez néha kilő persze olyan ütemezéseket is, amik (ha megnéznénk a belső számolásokat is, akkor) sorosíthatók lennének, de ez nem baj.

Példák

A korábban látott két ütemezés átírva úgy, hogy a számolások ne látszódjanak:

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
r(B)	
w(B)	
	r(B)
	w(B)

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
	r(B)
	w(B)
r(B)	
w(B)	

r(A) jelentése beolvassuk A-t; w(A) jelentése kiírjuk A-t

Példák

A korábban látott két ütemezés átírva úgy, hogy a számolások ne látszódjanak:

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
r(B)	
w(B)	
	r(B)
	w(B)

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
	r(B)
	w(B)
r(B)	
w(B)	

r(A) jelentése beolvassuk A-t; w(A) jelentése kiírjuk A-t

Látszik, hogy az első esetben bármilyen számolást is csinálnak a tranzakciók a beolvasott adattal a kiírás előtt, a számolástől függetlenül ugyanaz lesz a hatás mint a $T_1\,T_2$ soros ütemezésnél.

Példák

A korábban látott két ütemezés átírva úgy, hogy a számolások ne látszódjanak:

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
r(B)	
w(B)	
	r(B)
	w(B)

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
	r(B)
	w(B)
r(B)	
w(B)	

r(A) jelentése beolvassuk A-t; w(A) jelentése kiírjuk A-t

Látszik, hogy az első esetben bármilyen számolást is csinálnak a tranzakciók a beolvasott adattal a kiírás előtt, a számolástől függetlenül ugyanaz lesz a hatás mint a $T_1\,T_2$ soros ütemezésnél.
A második esetben, ahogy már láttuk is, lehetséges olyan számolás, ami esetén nem lesz azonos a hatás semelyik sorossal, így ez a csak írásokat és olvasásokat tartalmazó ütemezés nem sorosítható.

Példák

A korábban látott két ütemezés átírva úgy, hogy a számolások ne látszódjanak:

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
r(B)	
w(B)	
	r(B)
	w(B)

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
	r(B)
	w(B)
r(B)	
w(B)	

r(A) jelentése beolvassuk A-t; w(A) jelentése kiírjuk A-t

Látszik, hogy az első esetben bármilyen számolást is csinálnak a tranzakciók a beolvasott adattal a kiírás előtt, a számolástől függetlenül ugyanaz lesz a hatás mint a $T_1\,T_2$ soros ütemezésnél.
A második esetben, ahogy már láttuk is, lehetséges olyan számolás, ami esetén nem lesz azonos a hatás semelyik sorossal, így ez a csak írásokat és olvasásokat tartalmazó ütemezés nem sorosítható. (Persze lehetséges olyan számolás, amivel kiegészítve sorosítható lenne, de most kegyetlenek vagyunk: ha van egy olyan, amivel rossz, akkor rossz.)

Jelölés: A táblázat helyett így fogjuk az ütemezéseket megadni (a két előbbi esetben például):

$r_1(A),\ w_1(A),\ r_2(A),\ w_2(A),\ r_1(B),\ w_1(B),\ r_2(B),\ w_2(B)$

illetve

$r_1(A),\ w_1(A),\ r_2(A),\ w_2(A),\ r_2(B),\ w_2(B),\ r_1(B),\ w_1(B)$

Feltevések még

Általános elv (ahogy az előbb már ki is derült), hogy inkább legyünk szigorúak és minősítsünk rossznak egy olyat, ami sorosítható lenne, ha jobban megnéznénk, mint hogy sorosíthatónak mondjunk egy olyat, ami esetleg nem az

Feltevések még

Általános elv (ahogy az előbb már ki is derült), hogy inkább legyünk szigorúak és minősítsünk rossznak egy olyat, ami sorosítható lenne, ha jobban megnéznénk, mint hogy sorosíthatónak mondjunk egy olyat, ami esetleg nem az $\Longrightarrow$ mindig egy erősebb feltételt fogunk tesztelni, aki ezt is túléli az biztos sorosítható.

Feltevések még

Általános elv (ahogy az előbb már ki is derült), hogy inkább legyünk szigorúak és minősítsünk rossznak egy olyat, ami sorosítható lenne, ha jobban megnéznénk, mint hogy sorosíthatónak mondjunk egy olyat, ami esetleg nem az $\Longrightarrow$ mindig egy erősebb feltételt fogunk tesztelni, aki ezt is túléli az biztos sorosítható.
Általában nem egy már adott ütemezésről kell eldönteni, hogy az sorosítható-e, hanem olyan technikákat, protokollokat használunk, amikkel elérjük, hogy csak sorosítható ütemezések jöjjenek létre.

Sorosíthatóság biztosítása zárakkal

Elve: A tranzakciók zárolják azokat az adatelemeket, amivel dolgoznak, és amíg valami zár alatt van, addig a többi tranzakció nem, vagy csak korlátozottan fér hozzá.

Sorosíthatóság biztosítása zárakkal

Elve: A tranzakciók zárolják azokat az adatelemeket, amivel dolgoznak, és amíg valami zár alatt van, addig a többi tranzakció nem, vagy csak korlátozottan fér hozzá.

Egyszerű tranzakciómodell

Csak egyféle zárkérés van (LOCK), mindegyik művelethez ezt a zárat kell megkapni. Ezen kívül van még zárelengedés (UNLOCK). Az ütemezésekben nem csak írás és olvasás lesz, hanem a zárkérések és zárelengedések is benne lesznek.

Sorosíthatóság biztosítása zárakkal

Elve: A tranzakciók zárolják azokat az adatelemeket, amivel dolgoznak, és amíg valami zár alatt van, addig a többi tranzakció nem, vagy csak korlátozottan fér hozzá.

Egyszerű tranzakciómodell

Csak egyféle zárkérés van (LOCK), mindegyik művelethez ezt a zárat kell megkapni. Ezen kívül van még zárelengedés (UNLOCK). Az ütemezésekben nem csak írás és olvasás lesz, hanem a zárkérések és zárelengedések is benne lesznek. Csak olyan zárkéréseket tartalmazó ütemezéseket akarunk majd megengedni, amik eleget tesznek néhány követelménynek.
A legális ütemezés jellemzői:

1. Az i-edik tranzakció, T_i, csak akkor olvashatja vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$) és a zárat még azóta nem engedte fel (nem volt még azóta $UNLOCK_i(A)$).

Sorosíthatóság biztosítása zárakkal

Elve: A tranzakciók zárolják azokat az adatelemeket, amivel dolgoznak, és amíg valami zár alatt van, addig a többi tranzakció nem, vagy csak korlátozottan fér hozzá.

Egyszerű tranzakciómodell

Csak egyféle zárkérés van (LOCK), mindegyik művelethez ezt a zárat kell megkapni. Ezen kívül van még zárelengedés (UNLOCK). Az ütemezésekben nem csak írás és olvasás lesz, hanem a zárkérések és zárelengedések is benne lesznek. Csak olyan zárkéréseket tartalmazó ütemezéseket akarunk majd megengedni, amik eleget tesznek néhány követelménynek.
A legális ütemezés jellemzői:

1. Az i-edik tranzakció, T_i, csak akkor olvashatja vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$) és a zárat még azóta nem engedte fel (nem volt még azóta $UNLOCK_i(A)$).
2. Ha T_i zárolja az A adategységet, akkor később valamikor el is kell engednie a zárat ($LOCK_i(A)$ után mindig van $UNLOCK_i(A)$).

Sorosíthatóság biztosítása zárakkal

Elve: A tranzakciók zárolják azokat az adatelemeket, amivel dolgoznak, és amíg valami zár alatt van, addig a többi tranzakció nem, vagy csak korlátozottan fér hozzá.

Egyszerű tranzakciómodell

Csak egyféle zárkérés van (LOCK), mindegyik művelethez ezt a zárat kell megkapni. Ezen kívül van még zárelengedés (UNLOCK). Az ütemezésekben nem csak írás és olvasás lesz, hanem a zárkérések és zárelengedések is benne lesznek. Csak olyan zárkéréseket tartalmazó ütemezéseket akarunk majd megengedni, amik eleget tesznek néhány követelménynek.
A legális ütemezés jellemzői:

1. Az i-edik tranzakció, T_i, csak akkor olvashatja vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$) és a zárat még azóta nem engedte fel (nem volt még azóta $UNLOCK_i(A)$).
2. Ha T_i zárolja az A adategységet, akkor később valamikor el is kell engednie a zárat ($LOCK_i(A)$ után mindig van $UNLOCK_i(A)$).
3. Egyszerre két különböző tranzakciónak nem lehet zárja ugyanazon az adategységen.

Példa

Példa legális zárkérésre, ütemezésre ebben a modellben:

$$l_1(A),\ \ r_1(A),\ \ w_1(A),\ \ u_1(A),\ \ l_2(A),\ \ r_2(A),\ \ w_2(A),\ \ u_2(A),$$

$$l_1(B),\ \ r_1(B),\ \ w_1(B),\ \ u_1(B),\ \ l_2(B),\ \ r_2(B),\ \ w_2(B),\ \ u_2(B),$$

Példa arra, hogy hogyan dolgozhat az ütemező azon az egyszerű tranzakciómodellben, hogy legális ütemezés alakuljon ki

Tegyük fel, hogy a következő sorrendben jönnek zárkérések és műveleti kérések az ütemezőhöz (két tranzakció van):

$l_1(A),\ \ r_1(A),\ \ w_1(A),\ \ l_1(B),\ \ u_1(A),\ \ l_2(A),\ \ r_2(A),\ \ w_2(A)$

Példa

Példa legális zárkérésre, ütemezésre ebben a modellben:

$$l_1(A),\ r_1(A),\ w_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A),\ u_2(A),$$

$$l_1(B),\ r_1(B),\ w_1(B),\ u_1(B),\ l_2(B),\ r_2(B),\ w_2(B),\ u_2(B),$$

Példa arra, hogy hogyan dolgozhat az ütemező azon az egyszerű tranzakciómodellben, hogy legális ütemezés alakuljon ki

Tegyük fel, hogy a következő sorrendben jönnek zárkérések és műveleti kérések az ütemezőhöz (két tranzakció van):

$l_1(A),\ r_1(A),\ w_1(A),\ l_1(B),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A)$

Eddig minden rendben van, minden kérést teljesíteni lehet. Ha azonban a további kérések

$l_2(B),\ u_2(A),\ r_2(B),\ w_2(B),\ u_2(B),\ r_1(B),\ w_1(B),\ u_1(B)$

Példa

Példa legális zárkérésre, ütemezésre ebben a modellben:

$$l_1(A),\ r_1(A),\ w_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A),\ u_2(A),$$

$$l_1(B),\ r_1(B),\ w_1(B),\ u_1(B),\ l_2(B),\ r_2(B),\ w_2(B),\ u_2(B),$$

Példa arra, hogy hogyan dolgozhat az ütemező azon az egyszerű tranzakciómodellben, hogy legális ütemezés alakuljon ki

Tegyük fel, hogy a következő sorrendben jönnek zárkérések és műveleti kérések az ütemezőhöz (két tranzakció van):

$l_1(A),\ r_1(A),\ w_1(A),\ l_1(B),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A)$

Eddig minden rendben van, minden kérést teljesíteni lehet. Ha azonban a további kérések

$l_2(B),\ u_2(A),\ r_2(B),\ w_2(B),\ u_2(B),\ r_1(B),\ w_1(B),\ u_1(B)$

akkor ez már így nem mehet, mert T_2 nem kaphatja meg a kért zárat B-n, hiszen T_1 még tartja.

Példa

Példa legális zárkérésre, ütemezésre ebben a modellben:

$$l_1(A),\ r_1(A),\ w_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A),\ u_2(A),$$

$$l_1(B),\ r_1(B),\ w_1(B),\ u_1(B),\ l_2(B),\ r_2(B),\ w_2(B),\ u_2(B),$$

Példa arra, hogy hogyan dolgozhat az ütemező azon az egyszerű tranzakciómodellben, hogy legális ütemezés alakuljon ki

Tegyük fel, hogy a következő sorrendben jönnek zárkérések és műveleti kérések az ütemezőhöz (két tranzakció van):

$l_1(A),\ r_1(A),\ w_1(A),\ l_1(B),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A)$

Eddig minden rendben van, minden kérést teljesíteni lehet. Ha azonban a további kérések

$l_2(B),\ u_2(A),\ r_2(B),\ w_2(B),\ u_2(B),\ r_1(B),\ w_1(B),\ u_1(B)$

akkor ez már így nem mehet, mert T_2 nem kaphatja meg a kért zárat B-n, hiszen T_1 még tartja.

Emiatt az ütemező késlelteti T_2-t (T_2 vár T_1-re) és előbb engedi futni T_1-et, aztán jöhet T_2:

Példa

Példa legális zárkérésre, ütemezésre ebben a modellben:

$$l_1(A),\ r_1(A),\ w_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A),\ u_2(A),$$

$$l_1(B),\ r_1(B),\ w_1(B),\ u_1(B),\ l_2(B),\ r_2(B),\ w_2(B),\ u_2(B),$$

Példa arra, hogy hogyan dolgozhat az ütemező azon az egyszerű tranzakciómodellben, hogy legális ütemezés alakuljon ki

Tegyük fel, hogy a következő sorrendben jönnek zárkérések és műveleti kérések az ütemezőhöz (két tranzakció van):

$l_1(A),\ r_1(A),\ w_1(A),\ l_1(B),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A)$

Eddig minden rendben van, minden kérést teljesíteni lehet. Ha azonban a további kérések

$l_2(B),\ u_2(A),\ r_2(B),\ w_2(B),\ u_2(B),\ r_1(B),\ w_1(B),\ u_1(B)$

akkor ez már így nem mehet, mert T_2 nem kaphatja meg a kért zárat B-n, hiszen T_1 még tartja.

Emiatt az ütemező késlelteti T_2-t (T_2 vár T_1-re) és előbb engedi futni T_1-et, aztán jöhet T_2:

$r_1(B),\ w_1(B),\ u_1(B),\ l_2(B),\ u_2(A),\ r_2(B),\ w_2(B),\ u_2(B)$

lesz az az ütemezés, ami le fog futni, ez már legális lesz.

Holtpont

Láttuk, hogy az ütemező úgy kényszeríti ki a legális ütemezést, hogy várakoztatja a tranzakciókat. Ebből problémák lehetnek, ha a tranzakciók egymásra várnak:

Holtpont

Láttuk, hogy az ütemező úgy kényszeríti ki a legális ütemezést, hogy várakoztatja a tranzakciókat. Ebből problémák lehetnek, ha a tranzakciók egymásra várnak:
Holtpont (deadlock, patt): néhány zárkérés után akkor van holtpont, ha van egy olyan részhalmaza a tranzakcióknak, akik közül egyik se tud tovább futni, mert vár egy szintén ebben a részhalmazban levő másikra (vár egy olyan zár elengedésére, amit egy másik, ebbe a részhalmazba tartozó, tranzakció tart).

Holtpont

Láttuk, hogy az ütemező úgy kényszeríti ki a legális ütemezést, hogy várakoztatja a tranzakciókat. Ebből problémák lehetnek, ha a tranzakciók egymásra várnak:
Holtpont (deadlock, patt): néhány zárkérés után akkor van holtpont, ha van egy olyan részhalmaza a tranzakcióknak, akik közül egyik se tud tovább futni, mert vár egy szintén ebben a részhalmazban levő másikra (vár egy olyan zár elengedésére, amit egy másik, ebbe a részhalmazba tartozó, tranzakció tart).
Például:
$l_1(A),\ l_2(B),\ l_3(C),\ l_1(B),\ l_2(C),\ l_3(A)$
sorrendben érkező zárkérések esetén egyik tranzakció se tud tovább futni.

Holtpont

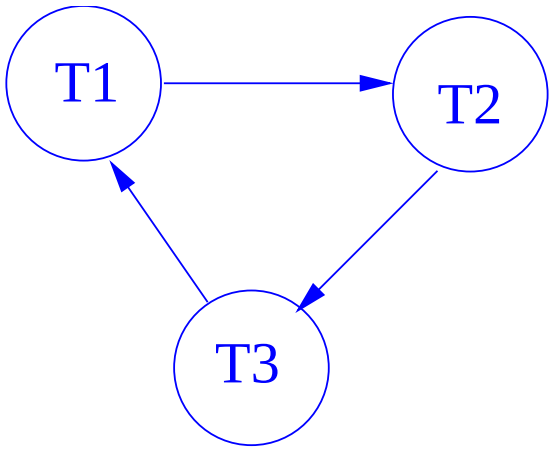
Láttuk, hogy az ütemező úgy kényszeríti ki a legális ütemezést, hogy várakoztatja a tranzakciókat. Ebből problémák lehetnek, ha a tranzakciók egymásra várnak:
Holtpont (deadlock, patt): néhány zárkérés után akkor van holtpont, ha van egy olyan részhalmaza a tranzakcióknak, akik közül egyik se tud tovább futni, mert vár egy szintén ebben a részhalmazban levő másikra (vár egy olyan zár elengedésére, amit egy másik, ebbe a részhalmazba tartozó, tranzakció tart).
Például:
$l_1(A),\ l_2(B),\ l_3(C),\ l_1(B),\ l_2(C),\ l_3(A)$
sorrendben érkező zárkérések esetén egyik tranzakció se tud tovább futni.
Az ilyen helyzeteket el kell kerülni, illetve ha már kialakultak, akkor fel kell ismerni és meg kell szüntetni.

Várakozási gráf

A felismerésben segít a zárkérések sorozatához tratozó **várakozási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha T_i vár egy olyan zár elengedésére, amit T_j tart éppen.

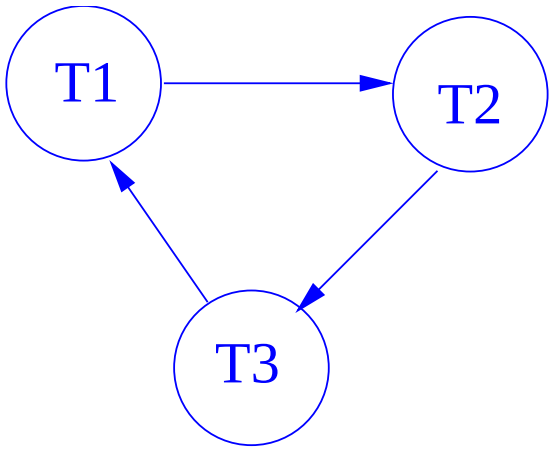
Várakozási gráf

A felismerésben segít a zárkérések sorozatához tratozó **várakozási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha T_i vár egy olyan zár elengedésére, amit T_j tart éppen.
Például az előbbi, holtponthoz vezető zárkéréssorozat várakozási gráfja a hat zárkérés után:



Várakozási gráf

A felismerésben segít a zárkérések sorozatához tratozó **várakozási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha T_i vár egy olyan zár elengedésére, amit T_j tart éppen.
Például az előbbi, holtponthoz vezető zárkéréssorozat várakozási gráfja a hat zárkérés után:



Vegyük észre, hogy a várakozási gráf változik az ütemezés során, ahogy újabb zárkérések érkeznek vagy zárelengedések történnek.

Holtpont felismerése

A várakozási gráf segítségével fel lehet ismerni a holtpontot az alábbi tétel miatt:

Holtpont felismerése

A várakozási gráf segítségével fel lehet ismerni a holtpontot az alábbi tétel miatt:

Tétel

Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráf DAG (nincs benne irányított kör).

Holtpont felismerése

A várakozási gráf segítségével fel lehet ismerni a holtpontot az alábbi tétel miatt:

Tétel

Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráf DAG (nincs benne irányított kör).

Bizonyítás.

$\Rightarrow$: Ha van irányított kör a várakozási gráfban, akkor a körbeli tranzakciók egyike se tud lefutni, mert vár a mellette levőre. Ez holtpont.

Holtpont felismerése

A várakozási gráf segítségével fel lehet ismerni a holtpontot az alábbi tétel miatt:

Tétel

Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráf DAG (nincs benne irányított kör).

Bizonyítás.

$\Rightarrow$: Ha van irányított kör a várakozási gráfban, akkor a körbeli tranzakciók egyike se tud lefutni, mert vár a mellette levőre. Ez holtpont.
$\Leftarrow$: Ha a gráf DAG, akkor van topológikus rendezése a tranzakcióknak és ebben a sorrendben le tudnak futni a tranzakciók. (Az első nem vár senkire, mert nem megy belőle ki él, így lefuthat; ezután már a másodikba se megy él, az is lefuthat . . .) □

Példa

Nézzük az alábbi írásokból és olvasásokból álló ütemezést:

$$r_1(A),\ \ r_2(B),\ \ w_1(C),\ \ r_3(D),\ \ r_4(E),\ \ r_3(B),\ \ w_2(C),\ \ w_4(A),\ \ w_1(D)$$

Példa

Nézzük az alábbi írásokból és olvasásokból álló ütemezést:

$$r_1(A),\ \ r_2(B),\ \ w_1(C),\ \ r_3(D),\ \ r_4(E),\ \ r_3(B),\ \ w_2(C),\ \ w_4(A),\ \ w_1(D)$$

Tegyük fel, hogy a zárkérések mindig közvetlenül megelőzik a műveletet, a zárelengedések pedig a tranzakciók végén, egyszerre történnek. Hogyan alakul a várakozási gráf ezen sorozat esetén? Lesz-e valamikor holtpont?

Példa

Nézzük az alábbi írásokból és olvasásokból álló ütemezést:

$$r_1(A),\ \ r_2(B),\ \ w_1(C),\ \ r_3(D),\ \ r_4(E),\ \ r_3(B),\ \ w_2(C),\ \ w_4(A),\ \ w_1(D)$$

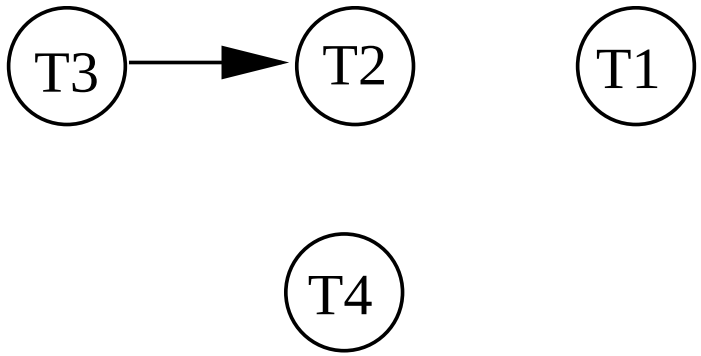
Tegyük fel, hogy a zárkérések mindig közvetlenül megelőzik a műveletet, a zárelengedések pedig a tranzakciók végén, egyszerre történnek. Hogyan alakul a várakozási gráf ezen sorozat esetén? Lesz-e valamikor holtpont?
Az elején $l_1(A),\ \ r_1(A),\ \ l_2(B),\ \ r_2(B),\ \ l_1(C),\ \ w_1(C),\ \ l_3(D),\ \ r_3(D),\ \ l_4(E),\ \ r_4(E)$ zárkérések és műveletek vannak, eddig még senki nem vár senkire.

Példa

Nézzük az alábbi írásokból és olvasásokból álló ütemezést:

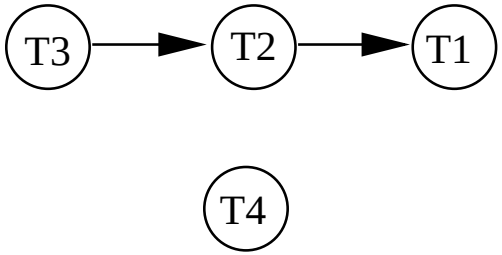
$$r_1(A),\ \ r_2(B),\ \ w_1(C),\ \ r_3(D),\ \ r_4(E),\ \ r_3(B),\ \ w_2(C),\ \ w_4(A),\ \ w_1(D)$$

Tegyük fel, hogy a zárkérések mindig közvetlenül megelőzik a műveletet, a zárelengedések pedig a tranzakciók végén, egyszerre történnek. Hogyan alakul a várakozási gráf ezen sorozat esetén? Lesz-e valamikor holtpont?
Az elején $l_1(A),\ \ r_1(A),\ \ l_2(B),\ \ r_2(B),\ \ l_1(C),\ \ w_1(C),\ \ l_3(D),\ \ r_3(D),\ \ l_4(E),\ \ r_4(E)$ zárkérések és műveletek vannak, eddig még senki nem vár senkire.
Ezután $l_3(B)$ jön $r_3(B)$ miatt, de T_3-nak várnia kell T_2-re:



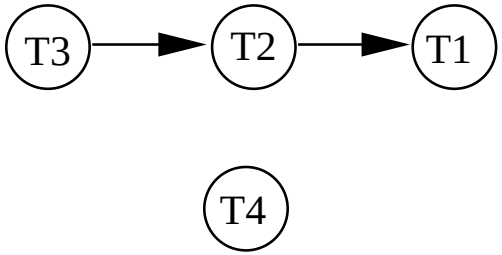
Példa

Ezután $l_2(C)$ jön $w_2(C)$ miatt, de T_2-nek is várnia kell T_1-re:

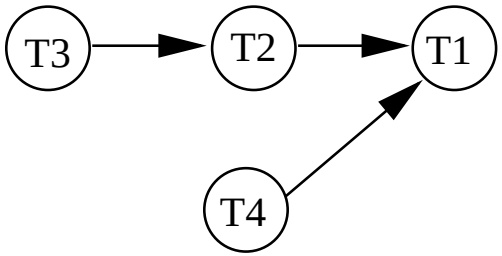


Példa

Ezután $l_2(C)$ jön $w_2(C)$ miatt, de T_2-nek is várnia kell T_1-re:

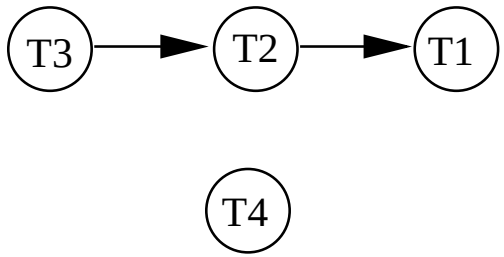


Ezután $l_4(A)$ jön $w_4(A)$ miatt, de T_4-nek is várnia kell T_1-re:

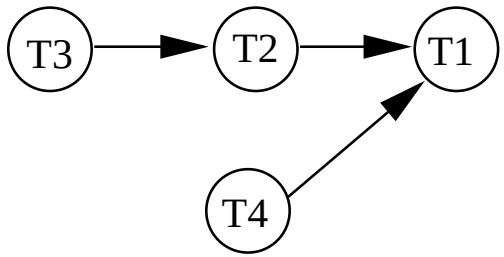


Példa

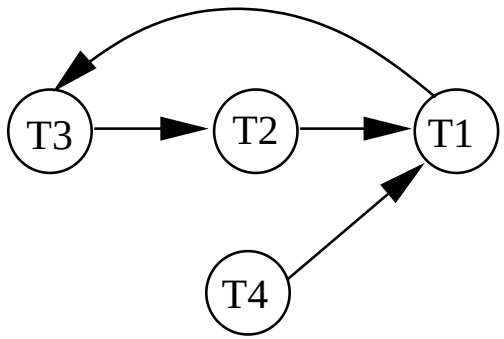
Ezután $l_2(C)$ jön $w_2(C)$ miatt, de T_2-nek is várnia kell T_1-re:



Ezután $l_4(A)$ jön $w_4(A)$ miatt, de T_4-nek is várnia kell T_1-re:

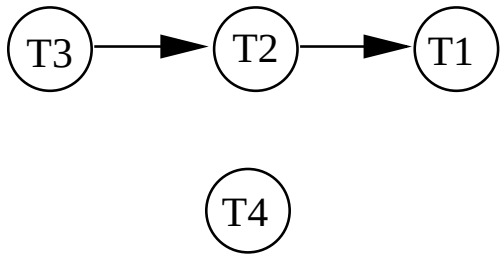


Végül $l_1(D)$ jön $w_1(D)$ miatt, de T_1-nek meg T_3-ra kell várnia:

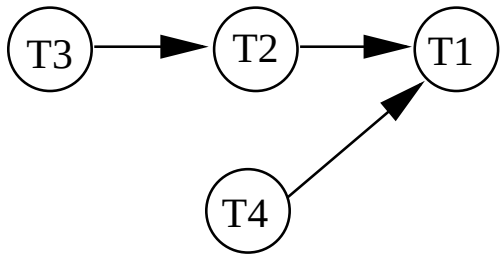


Példa

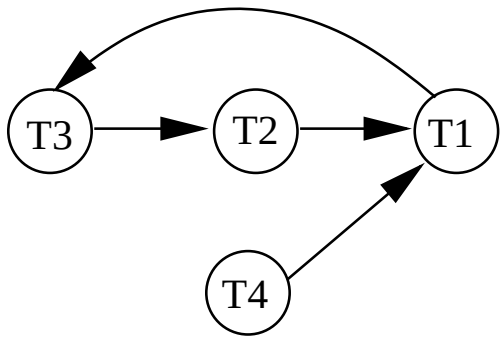
Ezután $l_2(C)$ jön $w_2(C)$ miatt, de T_2-nek is várnia kell T_1-re:



Ezután $l_4(A)$ jön $w_4(A)$ miatt, de T_4-nek is várnia kell T_1-re:



Végül $l_1(D)$ jön $w_1(D)$ miatt, de T_1-nek meg T_3-ra kell várnia:



És ez már holtpont: van irányított kör a gráfban.

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben.

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben.
Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_3, majd T_1 és T_4 is.

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben. Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_3, majd T_1 és T_4 is.
2. Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben. Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_3, majd T_1 és T_4 is.
2. Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:
   (a) Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog. Ha nem kapja meg az összeset, akkor egyet se kér el, el se indul.

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben. Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_3, majd T_1 és T_4 is.
2. Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:

(a) Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog. Ha nem kapja meg az összeset, akkor egyet se kér el, el se indul.
Ilyenkor biztos nem lesz holtpont, mert ha valaki megkap egy zárat, akkor le is tud futni, nem akad el. Az csak a baj ezzel, hogy előre kell mindent tudni.

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben. Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_3, majd T_1 és T_4 is.
2. Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:

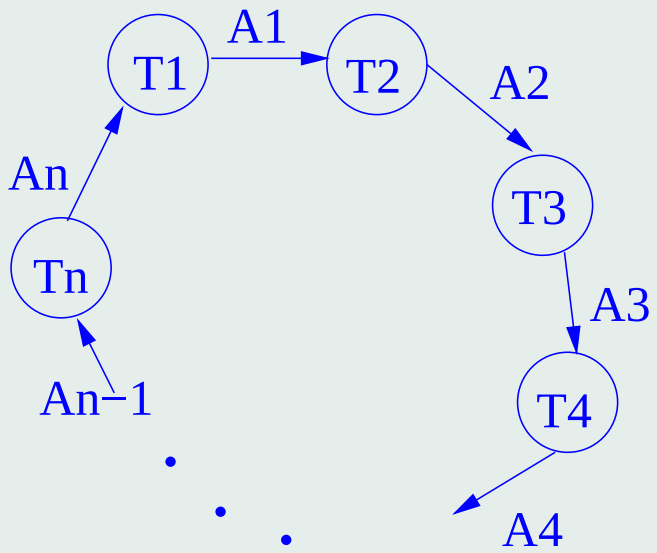
(a) Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog. Ha nem kapja meg az összeset, akkor egyet se kér el, el se indul.
Ilyenkor biztos nem lesz holtpont, mert ha valaki megkap egy zárat, akkor le is tud futni, nem akad el. Az csak a baj ezzel, hogy előre kell mindent tudni.

(b) Feltesszük, hogy van egy sorrend az adategységeken és minden egyes tranzakció csak eszerint a sorrend szerint növőleg kérhet újabb zárakat, azaz ha egy adategységre kért már zárat, akkor kisebb sorszámúra már nem kérhet később. Itt lehet, hogy lesz várakozás, de holtpont biztos nem lesz.

Megoldások holtpont ellen

Bizonyítás.

Ha valamely pillanatban lenne irányított kör a várakozási gráfban:

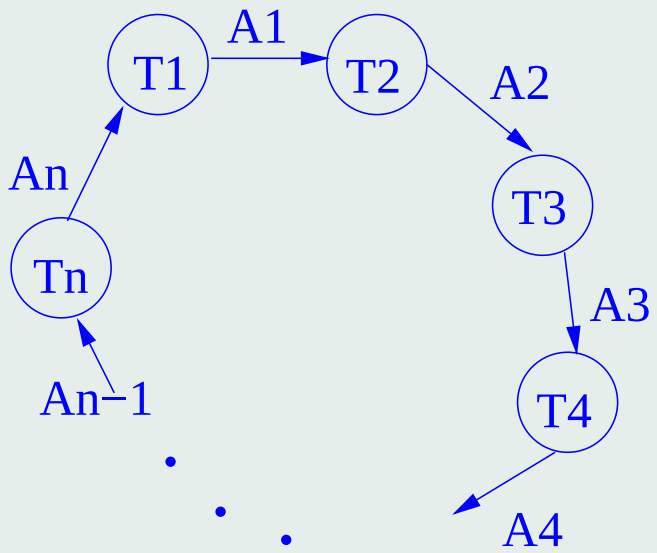


ahol T_i vár T_{i+1}-re az A_i adategység miatt, akkor $A_1 < A_2 < A_3 < \ldots A_n < A_1$ áll fenn abban az esetben, ha mindegyik tranzakció betartotta, hogy növőleg kér zárat.

Megoldások holtpont ellen

Bizonyítás.

Ha valamely pillanatban lenne irányított kör a várakozási gráfban:

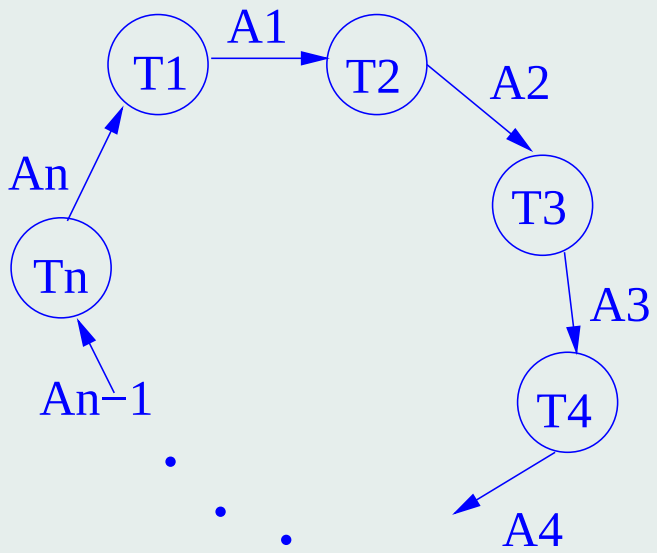


ahol T_i vár T_{i+1}-re az A_i adategység miatt, akkor $A_1 < A_2 < A_3 < \ldots A_n < A_1$ áll fenn abban az esetben, ha mindegyik tranzakció betartotta, hogy növőleg kér zárat. Ez azonban ellentmondás.

Megoldások holtpont ellen

Bizonyítás.

Ha valamely pillanatban lenne irányított kör a várakozási gráfban:



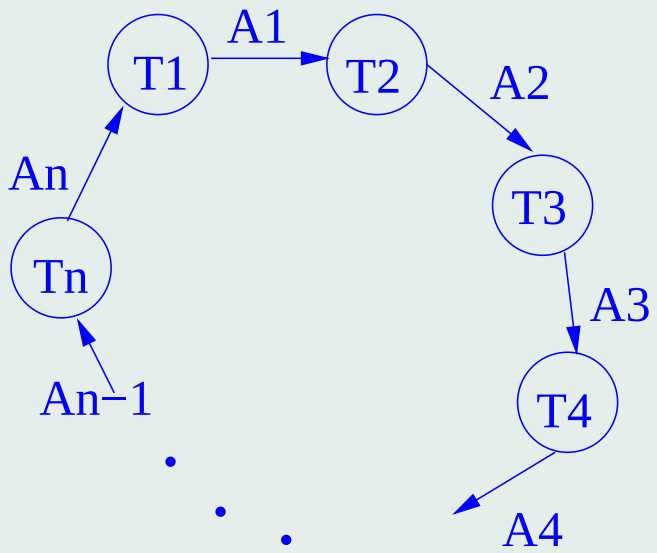
ahol T_i vár T_{i+1}-re az A_i adategység miatt, akkor $A_1 < A_2 < A_3 < \ldots A_n < A_1$ áll fenn abban az esetben, ha mindegyik tranzakció betartotta, hogy növőleg kér zárat. Ez azonban ellentmondás. □

Tehát ez a protokoll is megelőzi a holtpontot, de itt is előre kell tudni, hogy milyen zárakat fog kérni egy tranzakció.

Megoldások holtpont ellen

Bizonyítás.

Ha valamely pillanatban lenne irányított kör a várakozási gráfban:



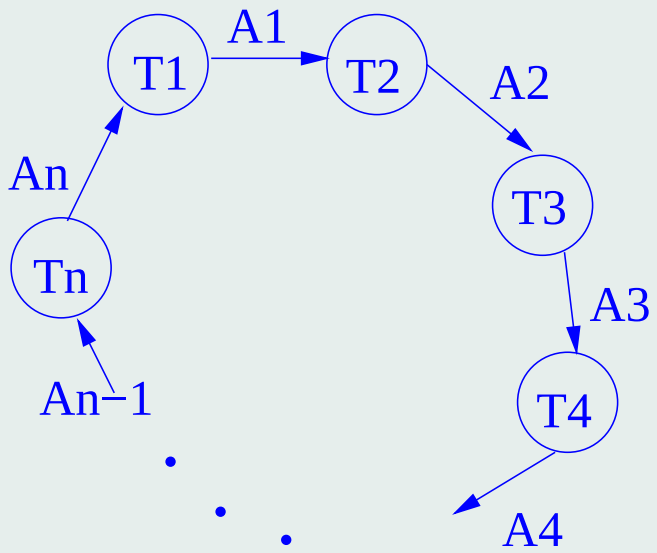
ahol T_i vár T_{i+1}-re az A_i adategység miatt, akkor $A_1 < A_2 < A_3 < \ldots A_n < A_1$ áll fenn abban az esetben, ha mindegyik tranzakció betartotta, hogy növőleg kér zárat. Ez azonban ellentmondás. □

Tehát ez a protokoll is megelőzi a holtpontot, de itt is előre kell tudni, hogy milyen zárakat fog kérni egy tranzakció.
Még egy módszer, ami szintén optimista, mint az első:

Megoldások holtpont ellen

Bizonyítás.

Ha valamely pillanatban lenne irányított kör a várakozási gráfban:



ahol T_i vár T_{i+1}-re az A_i adategység miatt, akkor $A_1 < A_2 < A_3 < \ldots A_n < A_1$ áll fenn abban az esetben, ha mindegyik tranzakció betartotta, hogy növőleg kér zárat. Ez azonban ellentmondás. □

Tehát ez a protokoll is megelőzi a holtpontot, de itt is előre kell tudni, hogy milyen zárakat fog kérni egy tranzakció.

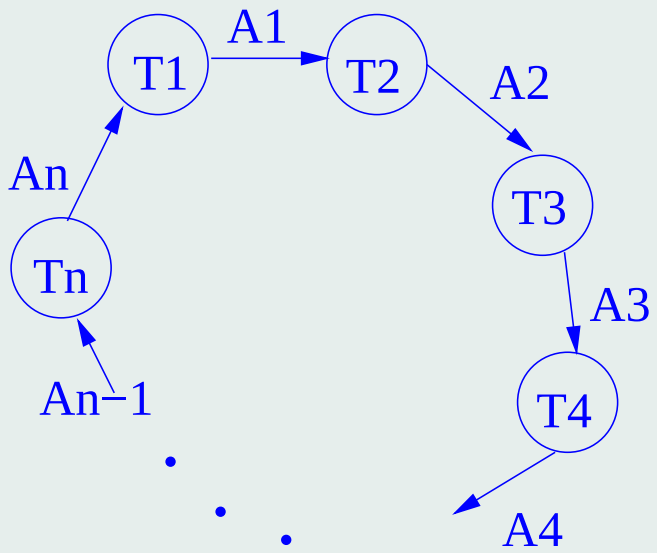
Még egy módszer, ami szintén optimista, mint az első:

Időkorlát alkalmazása: ha egy tranzakció kezdete óta túl sok idő telt el: ABORT.

Megoldások holtpont ellen

Bizonyítás.

Ha valamely pillanatban lenne irányított kör a várakozási gráfban:



ahol T_i vár T_{i+1}-re az A_i adategység miatt, akkor $A_1 < A_2 < A_3 < \ldots A_n < A_1$ áll fenn abban az esetben, ha mindegyik tranzakció betartotta, hogy növőleg kér zárat. Ez azonban ellentmondás. □

Tehát ez a protokoll is megelőzi a holtpontot, de itt is előre kell tudni, hogy milyen zárakat fog kérni egy tranzakció.

Még egy módszer, ami szintén optimista, mint az első:

Időkorlát alkalmazása: ha egy tranzakció kezdete óta túl sok idő telt el: ABORT.

Ehhez az kell, hogy ezt az időkorlátot jól tudjuk megválasztani.

Adatbázisok elmélete

Zárak alkalmazása

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

23. előadás

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.
Megoldás: adategységenként FIFO listában tartani a várakozókat

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.
Megoldás: adategységenként FIFO listában tartani a várakozókat
Eddig azt láttuk csak, hogy mennyi baj lehet a zárak alkalmazásával (holtpont, éhezés). Most nézzük, hogy mire jók a zárak.

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.
Megoldás: adategységenként FIFO listában tartani a várakozókat
Eddig azt láttuk csak, hogy mennyi baj lehet a zárak alkalmazásával (holtpont, éhezés). Most nézzük, hogy mire jók a zárak.
A zárak segítségével el lehet majd érni, hogy az ütemezések sorosíthatók legyenek, de pusztán az, hogy használjuk a zárakat, még nem ad sorosítható ütemezést.

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.
Megoldás: adategységenként FIFO listában tartani a várakozókat
Eddig azt láttuk csak, hogy mennyi baj lehet a zárak alkalmazásával (holtpont, éhezés). Most nézzük, hogy mire jók a zárak.
A zárak segítségével el lehet majd érni, hogy az ütemezések sorosíthatók legyenek, de pusztán az, hogy használjuk a zárakat, még nem ad sorosítható ütemezést.
Példa olyan legális, zárakat használó ütemezésre, ami nem sorosítható:

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.
Megoldás: adategységenként FIFO listában tartani a várakozókat
Eddig azt láttuk csak, hogy mennyi baj lehet a zárak alkalmazásával (holtpont, éhezés). Most nézzük, hogy mire jók a zárak.
A zárak segítségével el lehet majd érni, hogy az ütemezések sorosíthatók legyenek, de pusztán az, hogy használjuk a zárakat, még nem ad sorosítható ütemezést.
Példa olyan legális, zárakat használó ütemezésre, ami nem sorosítható: a korábbi, nem sorosítható, írásokból és olvasásokból álló ütemezésbe zárakat rakunk:

$$l_1(A),\ r_1(A),\ w_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A),\ u_2(A),$$
$$l_2(B),\ r_2(B),\ w_2(B),\ u_2(B),\ l_1(B),\ r_1(B),\ w_1(B),\ u_1(B)$$

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.
Egyszerű tranzakció modellben vagyunk (egy fajta zár van csak és a korábbi három feltevés mindig fennáll, azaz a zárkérés legális) és még valamit felteszünk:

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.
Egyszerű tranzakció modellben vagyunk (egy fajta zár van csak és a korábbi három feltevés mindig fennáll, azaz a zárkérés legális) és még valamit felteszünk:
A sorosíthatóságról pusztán a zárkérések alapján fogunk dönteni, nem nézzük azt, hogy ezeken kívül milyen műveletek (írások/olvasások) vannak. Pontosabban:

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.
Egyszerű tranzakció modellben vagyunk (egy fajta zár van csak és a korábbi három feltevés mindig fennáll, azaz a zárkérés legális) és még valamit felteszünk:
A sorosíthatóságról pusztán a zárkérések alapján fogunk dönteni, nem nézzük azt, hogy ezeken kívül milyen műveletek (írások/olvasások) vannak. Pontosabban:
Nem foglalkozunk azzal, hogy $LOCK_i(A)$ és $UNLOCK_i(A)$ között mi történik, feltesszük hogy valami teljesen egyedi írás és olvasás is.

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.
Egyszerű tranzakció modellben vagyunk (egy fajta zár van csak és a korábbi három feltevés mindig fennáll, azaz a zárkérés legális) és még valamit felteszünk:
A sorosíthatóságról pusztán a zárkérések alapján fogunk dönteni, nem nézzük azt, hogy ezeken kívül milyen műveletek (írások/olvasások) vannak. Pontosabban:
Nem foglalkozunk azzal, hogy $LOCK_i(A)$ és $UNLOCK_i(A)$ között mi történik, feltesszük hogy valami teljesen egyedi írás és olvasás is. Ez hasonlít ahhoz a helyzethez, mint amikor a konkrét számolásokat elhanyagoltuk: feltesszük, hogy a lehető legrosszabb történik azalatt, amíg a tranzakciónál van a zár.

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.
Egyszerű tranzakció modellben vagyunk (egy fajta zár van csak és a korábbi három feltevés mindig fennáll, azaz a zárkérés legális) és még valamit felteszünk:
A sorosíthatóságról pusztán a zárkérések alapján fogunk dönteni, nem nézzük azt, hogy ezeken kívül milyen műveletek (írások/olvasások) vannak. Pontosabban:
Nem foglalkozunk azzal, hogy $LOCK_i(A)$ és $UNLOCK_i(A)$ között mi történik, feltesszük hogy valami teljesen egyedi írás és olvasás is. Ez hasonlít ahhoz a helyzethez, mint amikor a konkrét számolásokat elhanyagoltuk: feltesszük, hogy a lehető legrosszabb történik azalatt, amíg a tranzakciónál van a zár.
Így persze megint igaz lesz az, hogy olyan ütemezéseket is rossznak minősítünk, amik igazából sorosíthatók lennének, ha megnéznénk, hogy írások vagy olvasások történnek, de ez nem baj, mert szigorúbbak lehetünk, csak az a fontos, hogy olyan ne legyen sorosíthatónak minősítve, aki nem az.

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.
Mikor lesz egy ilyen ütemezés sorosítható, függetlenül attól, hogy milyen írások és olvasások történnek valójában?

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.
Mikor lesz egy ilyen ütemezés sorosítható, függetlenül attól, hogy milyen írások és olvasások történnek valójában?
Ennek megválaszolásában segít a **sorosítási gráf**:

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.
Mikor lesz egy ilyen ütemezés sorosítható, függetlenül attól, hogy milyen írások és olvasások történnek valójában?
Ennek megválaszolásában segít a **sorosítási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha az ütemezésben van olyan $u_i(A) \quad \ldots \quad l_j(A)$ rész, ahol $u_i(A)$ (T_i elengedi A zárját) és $l_j(A)$ (T_j megkapja A zárját) között A-ra senki se kap zárat.

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.
Mikor lesz egy ilyen ütemezés sorosítható, függetlenül attól, hogy milyen írások és olvasások történnek valójában?
Ennek megválaszolásában segít a **sorosítási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha az ütemezésben van olyan $u_i(A) \quad \ldots \quad l_j(A)$ rész, ahol $u_i(A)$ (T_i elengedi A zárját) és $l_j(A)$ (T_j megkapja A zárját) között A-ra senki se kap zárat.
Ekkor minden olyan soros ütemezésben, ami ekvivalens lehet a miénkkel, biztos, hogy T_j-nek T_i után kell jönnie.

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.
Mikor lesz egy ilyen ütemezés sorosítható, függetlenül attól, hogy milyen írások és olvasások történnek valójában?
Ennek megválaszolásában segít a **sorosítási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha az ütemezésben van olyan $u_i(A) \quad \ldots \quad l_j(A)$ rész, ahol $u_i(A)$ (T_i elengedi A zárját) és $l_j(A)$ (T_j megkapja A zárját) között A-ra senki se kap zárat.
Ekkor minden olyan soros ütemezésben, ami ekvivalens lehet a miénkkel, biztos, hogy T_j-nek T_i után kell jönnie.
Ez azért van így, mert feltettük, hogy T_i is és T_j is bármit csinálhat A-val, amíg nála van a zár és ha pl. T_i írja, T_j meg olvassa A-t, akkor már csak a $T_i, \ldots, T_j$ sorrend lesz a jó.

Példa sorosítási gráfra

Az alábbi, csak zárkéréseket és zárelengedéseket tartalmazó ütemezés legális (HF: leellenőrizni):

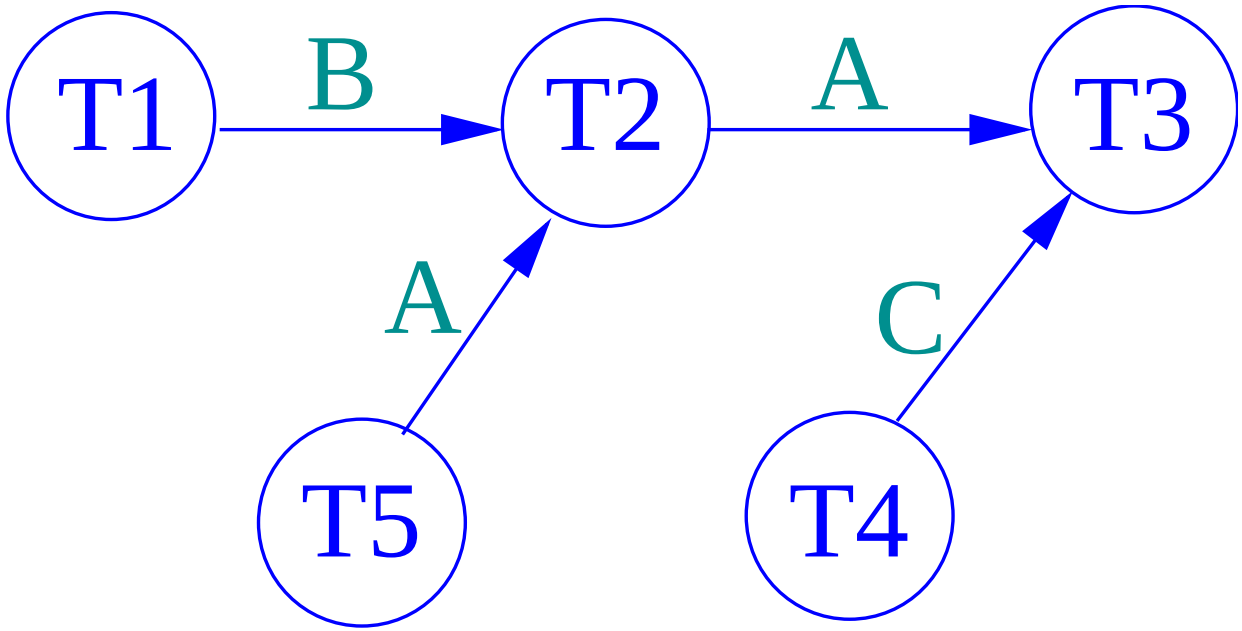
$$l_5(A),\ \ l_1(B),\ \ u_5(A),\ \ l_4(C),\ \ u_1(B),\ \ l_2(A),\ \ l_2(B),\ \ u_2(A),$$
$$l_3(A),\ \ u_3(A),\ \ u_4(C),\ \ u_2(B),\ \ l_3(C),\ \ u_3(C)$$

Példa sorosítási gráfra

Az alábbi, csak zárkéréseket és zárelengedéseket tartalmazó ütemezés legális (HF: leellenőrizni):

$$l_5(A),\ l_1(B),\ u_5(A),\ l_4(C),\ u_1(B),\ l_2(A),\ l_2(B),\ u_2(A),$$
$$l_3(A),\ u_3(A),\ u_4(C),\ u_2(B),\ l_3(C),\ u_3(C)$$

Az ehhez tartozó sorosítási gráf:



Tétel a sorosítási gráfról

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Tétel a sorosítási gráfról

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Bizonyítás.

$\Rightarrow$: Ha nem DAG a gráf, akkor van benne irányított kör. Ebben a körben levő tranzakciók közül egyik sem előzheti meg a többit egy ekvivalens soros ütemezésben, amiből következik, hogy nincs ekvivalens soros ütemezés.

Tétel a sorosítási gráfról

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Bizonyítás.

$\Rightarrow$: Ha nem DAG a gráf, akkor van benne irányított kör. Ebben a körben levő tranzakciók közül egyik sem előzheti meg a többit egy ekvivalens soros ütemezésben, amiből következik, hogy nincs ekvivalens soros ütemezés.
$\Leftarrow$: Teljes indukcióval: $n = 1$-re (1 tranzakció van csak) világos, egy ilyen ütemezés maga soros.

Tétel a sorosítási gráfról

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Bizonyítás.

$\Rightarrow$: Ha nem DAG a gráf, akkor van benne irányított kör. Ebben a körben levő tranzakciók közül egyik sem előzheti meg a többit egy ekvivalens soros ütemezésben, amiből következik, hogy nincs ekvivalens soros ütemezés.
$\Leftarrow$: Teljes indukcióval: $n = 1$-re (1 tranzakció van csak) világos, egy ilyen ütemezés maga soros.
Legyen most az ütemezésben n tranzakció. Ha a gráf DAG, akkor létezik topologikus rendezése. Azt látjuk be, hogy a topologikus sorrend szerinti soros ütemezés ekvivalens lesz az eredeti ütemezéssel.

Tétel a sorosítási gráfról

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Bizonyítás.

$\Rightarrow$: Ha nem DAG a gráf, akkor van benne irányított kör. Ebben a körben levő tranzakciók közül egyik sem előzheti meg a többit egy ekvivalens soros ütemezésben, amiből következik, hogy nincs ekvivalens soros ütemezés.
$\Leftarrow$: Teljes indukcióval: $n = 1$-re (1 tranzakció van csak) világos, egy ilyen ütemezés maga soros.
Legyen most az ütemezésben n tranzakció. Ha a gráf DAG, akkor létezik topologikus rendezése. Azt látjuk be, hogy a topologikus sorrend szerinti soros ütemezés ekvivalens lesz az eredeti ütemezéssel.
Ha T_i a topologikus rendezés szerinti első tranzakció, akkor nem megy bele él, vagyis ő csak olyan adategységeket használ, amiket az eredeti ütemezésben előtte senki. Így az ő összes utasítását előre mozgathatjuk, a hatás nem változik.

Tétel a sorosítási gráfról

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Bizonyítás.

$\Rightarrow$: Ha nem DAG a gráf, akkor van benne irányított kör. Ebben a körben levő tranzakciók közül egyik sem előzheti meg a többit egy ekvivalens soros ütemezésben, amiből következik, hogy nincs ekvivalens soros ütemezés.
$\Leftarrow$: Teljes indukcióval: $n = 1$-re (1 tranzakció van csak) világos, egy ilyen ütemezés maga soros.
Legyen most az ütemezésben n tranzakció. Ha a gráf DAG, akkor létezik topologikus rendezése. Azt látjuk be, hogy a topologikus sorrend szerinti soros ütemezés ekvivalens lesz az eredeti ütemezéssel.
Ha T_i a topologikus rendezés szerinti első tranzakció, akkor nem megy bele él, vagyis ő csak olyan adategységeket használ, amiket az eredeti ütemezésben előtte senki. Így az ő összes utasítását előre mozgathatjuk, a hatás nem változik.
Ami ezután marad, az $n - 1$ tranzakció utasításaiból álló ütemezés, aminek a sorosítási gráfja szintén DAG, tehát ennek az indukció szerint létezik soros ekvivalense (a maradék tranzakciók topologikus sorrendjének megfelelően), ami T_i-vel kiegészítve soros ekvivalense lesz az eredetinek. □

Következmény

Következmény: A bizonyításból látszik, hogy a soros ekvivalensek és a lehetséges topologikus sorrendek megfelelnek egymásnak, vagyis annyi soros ekvivalens lesz, ahány különböző topologikus sorrend van.

Következmény

Következmény: A bizonyításból látszik, hogy a soros ekvivalensek és a lehetséges topologikus sorrendek megfelelnek egymásnak, vagyis annyi soros ekvivalens lesz, ahány különböző topologikus sorrend van.
Például a korábban látott sorosítási gráf esetén 8 darab topologikus sorrend van, így nyolc soros ekvivalens van:
$T_5\ T_4\ T_1\ T_2\ T_3$,
$T_4\ T_5\ T_1\ T_2\ T_3$,
$T_4\ T_1\ T_5\ T_2\ T_3$,
$T_5\ T_1\ T_4\ T_2\ T_3$,
$T_1\ T_5\ T_4\ T_2\ T_3$,
$T_1\ T_4\ T_5\ T_2\ T_3$,
$T_5\ T_1\ T_2\ T_4\ T_3$,
$T_1\ T_5\ T_2\ T_4\ T_3$,

Példa, ami mutatja, hogy szigorúbbak vagyunk a kelleténél

Tekintsük az

$$l_1(A),\ r_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ u_2(A),\ l_1(A),\ w_1(A),\ u_1(A),\ l_2(B),\ r_2(B),\ u_2(B)$$

ütemezést.

Példa, ami mutatja, hogy szigorúbbak vagyunk a kelleténél

Tekintsük az

$$l_1(A),\ r_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ u_2(A),\ l_1(A),\ w_1(A),\ u_1(A),\ l_2(B),\ r_2(B),\ u_2(B)$$

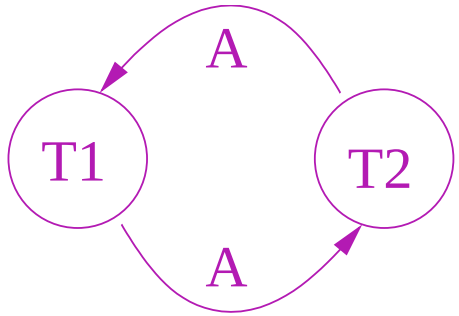
ütemezést.
Ha megnézzük az írás/olvasás műveleteket $(r_1(A),\ r_2(A),\ w_1(A),\ r_2(B))$, akkor látszik, hogy az ütemezés hatása azonos a $T_2 T_1$ soros ütemezés hatásával, vagyis ez egy sorosítható ütemezés.

Példa, ami mutatja, hogy szigorúbbak vagyunk a kelleténél

Tekintsük az

$$l_1(A),\ r_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ u_2(A),\ l_1(A),\ w_1(A),\ u_1(A),\ l_2(B),\ r_2(B),\ u_2(B)$$

ütemezést.
Ha megnézzük az írás/olvasás műveleteket $(r_1(A),\ r_2(A),\ w_1(A),\ r_2(B))$, akkor látszik, hogy az ütemezés hatása azonos a T_2T_1 soros ütemezés hatásával, vagyis ez egy sorosítható ütemezés.
De ha felrajzoljuk a sorosítási gráfot (és ilyenkor persze nem nézzük, hogy milyen írások/olvasások vannak, hanem a legrosszabb esetre készülünk), akkor



lesz a gráf, és mivel ez nem DAG, ezért nem lesz sorosítható az az ütemezés, amiben már csak a zárak vannak benne.

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
   (Előny: nem óvatoskodik, nem korlátoz feleslegesen;

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
(Előny: nem óvatoskodik, nem korlátoz feleslegesen;
Hátrány: drasztikus megoldás az ABORT)

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
   (Előny: nem óvatoskodik, nem korlátoz feleslegesen;
   Hátrány: drasztikus megoldás az ABORT)
2. Protokollt ír elő a tranzakciók számára, amit minden egyes tranzakciónak be kell tartania:

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
   (Előny: nem óvatoskodik, nem korlátoz feleslegesen;
   Hátrány: drasztikus megoldás az ABORT)
2. Protokollt ír elő a tranzakciók számára, amit minden egyes tranzakciónak be kell tartania:
   2PL (two-phase locking, kétfázisú protokoll): a T_i tranzakció követi a kétfázisú protokollt, ha $UNLOCK_i$ után nincs $LOCK_i$, azaz ha nem kér már zárat miután elengedett már egyet.

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
   (Előny: nem óvatoskodik, nem korlátoz feleslegesen;
   Hátrány: drasztikus megoldás az ABORT)
2. Protokollt ír elő a tranzakciók számára, amit minden egyes tranzakciónak be kell tartania:
   2PL (two-phase locking, kétfázisú protokoll): a T_i tranzakció követi a kétfázisú protokollt, ha $UNLOCK_i$ után nincs $LOCK_i$, azaz ha nem kér már zárat miután elengedett már egyet.

Tétel

Ha az egyszerű tranzakciómodellbeli legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
   (Előny: nem óvatoskodik, nem korlátoz feleslegesen;
   Hátrány: drasztikus megoldás az ABORT)
2. Protokollt ír elő a tranzakciók számára, amit minden egyes tranzakciónak be kell tartania:
   2PL (two-phase locking, kétfázisú protokoll): a T_i tranzakció követi a kétfázisú protokollt, ha $UNLOCK_i$ után nincs $LOCK_i$, azaz ha nem kér már zárat miután elengedett már egyet.

Tétel

Ha az egyszerű tranzakciómodellbeli legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.

Nem bizonyítjuk.

Bonyolultabb zármodellek

Többfajta zár van, aszerint, hogy a tranzakciók mit akarnak csinálni az adattal. (Persze akkor, ha van több különböző művelet, nem csak írás és olvasás.)

Bonyolultabb zármodellek

Többfajta zár van, aszerint, hogy a tranzakciók mit akarnak csinálni az adattal. (Persze akkor, ha van több különböző művelet, nem csak írás és olvasás.)
Cél, hogy minél jobban tükrözzék a zárkérési lehetőségek a lehetséges műveleteket, mert így kevesebb lesz a várakozás (több olyan eset lesz, amikor lehet két tranzakciónak zárja ugyanott, ha a megfelelő műveletek mehetnek együtt) és megalapozottabb lesz a döntés a sorosíthatóságról (nem leszünk annyira feleslegesen szigorúak).

Bonyolultabb zármodellek

Többfajta zár van, aszerint, hogy a tranzakciók mit akarnak csinálni az adattal. (Persze akkor, ha van több különböző művelet, nem csak írás és olvasás.)
Cél, hogy minél jobban tükrözzék a zárkérési lehetőségek a lehetséges műveleteket, mert így kevesebb lesz a várakozás (több olyan eset lesz, amikor lehet két tranzakciónak zárja ugyanott, ha a megfelelő műveletek mehetnek együtt) és megalapozottabb lesz a döntés a sorosíthatóságról (nem leszünk annyira feleslegesen szigorúak).
Példa: Legyen három művelet: olvasás, írás és növelés (increment).
Ez utóbbi azt jelenti, hogy az adategység aktuális értékét megnöveljük eggyel.

Bonyolultabb zármodellek

Többfajta zár van, aszerint, hogy a tranzakciók mit akarnak csinálni az adattal. (Persze akkor, ha van több különböző művelet, nem csak írás és olvasás.)
Cél, hogy minél jobban tükrözzék a zárkérési lehetőségek a lehetséges műveleteket, mert így kevesebb lesz a várakozás (több olyan eset lesz, amikor lehet két tranzakciónak zárja ugyanott, ha a megfelelő műveletek mehetnek együtt) és megalapozottabb lesz a döntés a sorosíthatóságról (nem leszünk annyira feleslegesen szigorúak).
Példa: Legyen három művelet: olvasás, írás és növelés (increment).
Ez utóbbi azt jelenti, hogy az adategység aktuális értékét megnöveljük eggyel.
Ekkor bevezethetünk három zárat: RLOCK, WLOCK és INCR, a kézenfekvő használattal (a megfelelő művelet csak akkor mehet, ha a tranzakció megkapta a hozzá tartozó zárat).

Kompatibilitási mátrix

Egy mátrix segítségével adjuk meg, hogy különböző tranzakcióknak milyen zárai lehetnek egyszerre egy adategységen.
Ez a **kompatibilitási mátrix**: a sorok és az oszlopok is a lehetséges záraknak felelnek meg és a Z_i sor Z_j oszlopában pontosan akkor van `I`, ha egy tranzakció megkaphatja egy adategységre a Z_j zárat akkor, ha egy másik tranzakció Z_i zárat tart fenn ezen az adategységen. Ha nem kaphatja meg, akkor `N` áll a Z_i sor Z_j oszlopában.

Kompatibilitási mátrix

Egy mátrix segítségével adjuk meg, hogy különböző tranzakcióknak milyen zárai lehetnek egyszerre egy adategységen.
Ez a **kompatibilitási mátrix**: a sorok és az oszlopok is a lehetséges záraknak felelnek meg és a Z_i sor Z_j oszlopában pontosan akkor van `I`, ha egy tranzakció megkaphatja egy adategységre a Z_j zárat akkor, ha egy másik tranzakció Z_i zárat tart fenn ezen az adategységen. Ha nem kaphatja meg, akkor `N` áll a Z_i sor Z_j oszlopában.
Akkor lehet két különböző tranzakciónak Z_i és Z_j zárja ugyanazon az adategységen, ha mindegy, hogy a két zárnak megfelelő műveletek milyen sorrendben hajtódnak végre.

Kompatibilitási mátrix

Egy mátrix segítségével adjuk meg, hogy különböző tranzakcióknak milyen zárai lehetnek egyszerre egy adategységen.
Ez a **kompatibilitási mátrix**: a sorok és az oszlopok is a lehetséges záraknak felelnek meg és a Z_i sor Z_j oszlopában pontosan akkor van `I`, ha egy tranzakció megkaphatja egy adategységre a Z_j zárat akkor, ha egy másik tranzakció Z_i zárat tart fenn ezen az adategységen. Ha nem kaphatja meg, akkor `N` áll a Z_i sor Z_j oszlopában.
Akkor lehet két különböző tranzakciónak Z_i és Z_j zárja ugyanazon az adategységen, ha mindegy, hogy a két zárnak megfelelő műveletek milyen sorrendben hajtódnak végre.
Ez alapján az RLOCK/WLOCK modell és az RLOCK/WLOCK/INCR modell mátrixai:

	RLOCK	WLOCK
RLOCK	I	N
WLOCK	N	N

	RLOCK	WLOCK	INCR
RLOCK	I	N	N
WLOCK	N	N	N
INCR	N	N	I

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az `I` a mátrixban, annál kevesebb lesz a várakoztatás.

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az `I` a mátrixban, annál kevesebb lesz a várakoztatás.
2. A mátrix alapján keletkező várakozásokhoz elkészített várakozási gráf segítségével az ütemező kezeli a holtpontot (ami tetszőleges zármodell esetén ugyanazt jelenti és a gráfot is ugyanúgy kell felépíteni).

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az `I` a mátrixban, annál kevesebb lesz a várakoztatás.
2. A mátrix alapján keletkező várakozásokhoz elkészített várakozási gráf segítségével az ütemező kezeli a holtpontot (ami tetszőleges zármodell esetén ugyanazt jelenti és a gráfot is ugyanúgy kell felépíteni).
3. A mátrix alapján készíti el az ütemező a sorosítási gráfot egy zárkérés-sorozathoz:

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az `I` a mátrixban, annál kevesebb lesz a várakoztatás.
2. A mátrix alapján keletkező várakozásokhoz elkészített várakozási gráf segítségével az ütemező kezeli a holtpontot (ami tetszőleges zármodell esetén ugyanazt jelenti és a gráfot is ugyanúgy kell felépíteni).
3. A mátrix alapján készíti el az ütemező a sorosítási gráfot egy zárkérés-sorozathoz: a sorosítási gráf csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha van olyan A adategység, amelyre az ütemezés során Z_k zárat kért és kapott T_i, ezt elengedte, majd ezután A-ra legközelebb T_j kért és kapott olyan Z_l zárat, hogy a mátrixban a Z_k sor Z_l oszlopában `N` áll.

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az `I` a mátrixban, annál kevesebb lesz a várakoztatás.
2. A mátrix alapján keletkező várakozásokhoz elkészített várakozási gráf segítségével az ütemező kezeli a holtpontot (ami tetszőleges zármodell esetén ugyanazt jelenti és a gráfot is ugyanúgy kell felépíteni).
3. A mátrix alapján készíti el az ütemező a sorosítási gráfot egy zárkérés-sorozathoz: a sorosítási gráf csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha van olyan A adategység, amelyre az ütemezés során Z_k zárat kért és kapott T_i, ezt elengedte, majd ezután A-ra legközelebb T_j kért és kapott olyan Z_l zárat, hogy a mátrixban a Z_k sor Z_l oszlopában `N` áll.
   Vagyis olyankor lesz él, ha a két zár nem kompatibilis egymással, nem mindegy a két művelet sorrendje.

Sorosíthatóság

A sorosíthatóságról most is pusztán a zárkérések alapján fogunk dönteni, a sorosítási gráf segítségével:

Sorosíthatóság

A sorosíthatóságról most is pusztán a zárkérések alapján fogunk dönteni, a sorosítási gráf segítségével:

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó legális ütemezés pontosan akkor sorosítható valamelyik zármodellben, ha a zármodellhez tartozó kompatibilitási mátrix alapján az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Sorosíthatóság

A sorosíthatóságról most is pusztán a zárkérések alapján fogunk dönteni, a sorosítási gráf segítségével:

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó legális ütemezés pontosan akkor sorosítható valamelyik zármodellben, ha a zármodellhez tartozó kompatibilitási mátrix alapján az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Bizonyítás.

Pontosan ugyanúgy megy, ahogyan eddig.

Sorosíthatóság

A sorosíthatóságról most is pusztán a zárkérések alapján fogunk dönteni, a sorosítási gráf segítségével:

Tétel

Egy csak zárkéréseket és zárelengedéseket tartalmazó legális ütemezés pontosan akkor sorosítható valamelyik zármodellben, ha a zármodellhez tartozó kompatibilitási mátrix alapján az előbbi módszerrel felrajzolt sorosítási gráf DAG.

Bizonyítás.

Pontosan ugyanúgy megy, ahogyan eddig. □

Az ütemező egyik lehetősége a sorosíthatóság elérésére, hogy folyamatosan figyeli a sorosítási gráfot és ha irányított kör keletezne, akkor ABORT-ot rendel el.

Sorosíthatóság II.

Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:

Tétel

Ha valamilyen zármodellben egy legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.

Sorosíthatóság II.

Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:

Tétel

Ha valamilyen zármodellben egy legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.

Sorosíthatóság II.

Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:

Tétel

Ha valamilyen zármodellben egy legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.

Nem bizonyítjuk.

Sorosíthatóság II.

Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:

Tétel

Ha valamilyen zármodellben egy legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.

Nem bizonyítjuk.
Megjegyzés: Minél gazdagabb a zármodell, minél több az `I` a kompatibilitási mátrixban, annál valószínűbb, hogy a sorosítási gráf DAG lesz minden külön protokoll nélkül. Ez azt jelenti, ilyenkor egyre jobb lesz az ABORT-os módszer (ritkán kellhet).

Mit látunk mi ebből?

Az adatbáziskezelő működése során az ütemező munkájába nem (nagyon) szólhatunk bele. Miért hasznos mégis tudni, hogy hogyan működik?

- Abba beleszólhatunk, hogy mennyire törekedjen sorosíthatóságra az adatbáziskezelő (akár azt is mondhatjuk, hogy semennyire). Ehhez nem árt, ha tudjuk, hogy mi is a sorosíthatóság, mit nyerünk vele és mibe kerül (bonyolult ütemező, lassabb futás).

Mit látunk mi ebből?

Az adatbáziskezelő működése során az ütemező munkájába nem (nagyon) szólhatunk bele. Miért hasznos mégis tudni, hogy hogyan működik?

- Abba beleszólhatunk, hogy mennyire törekedjen sorosíthatóságra az adatbáziskezelő (akár azt is mondhatjuk, hogy semennyire). Ehhez nem árt, ha tudjuk, hogy mi is a sorosíthatóság, mit nyerünk vele és mibe kerül (bonyolult ütemező, lassabb futás).
- Ha ismerjük a különféle ütemezési technikákat: jobban fogjuk érteni az előforduló ABORT-okat, és majd az ABORT utáni visszaállítást is.

Összefüggések az adategységek között

Eddig nem néztük azt, hogy mik azok az adategységek, amikre a zárakat lehet kérni és kapni. Hallgatólagosan feltettük, hogy ezeket egymástól függetlenül lehet zárolni, nincs közöttük semmi szervezettség, semmi összefüggés.

Összefüggések az adategységek között

Eddig nem néztük azt, hogy mik azok az adategységek, amikre a zárakat lehet kérni és kapni. Hallgatólagosan feltettük, hogy ezeket egymástól függetlenül lehet zárolni, nincs közöttük semmi szervezettség, semmi összefüggés.
A valóságban két különböző esetben sem alkalmazható ez a megközelítés:

1. Ha az adategységek egymásba ágyazottak (pl. reláció, blokk, rekord), ekkor még további megkötéseket szeretnénk a zárolásra, az eddigi módszereket ki kell egészíteni.

Összefüggések az adategységek között

Eddig nem néztük azt, hogy mik azok az adategységek, amikre a zárakat lehet kérni és kapni. Hallgatólagosan feltettük, hogy ezeket egymástól függetlenül lehet zárolni, nincs közöttük semmi szervezettség, semmi összefüggés.
A valóságban két különböző esetben sem alkalmazható ez a megközelítés:

1. Ha az adategységek egymásba ágyazottak (pl. reláció, blokk, rekord), ekkor még további megkötéseket szeretnénk a zárolásra, az eddigi módszereket ki kell egészíteni.
2. Ha tudjuk, hogy egymáshoz képest hogyan helyezkednek el az adategységek a tárolási struktúrában, akkor jobb módszereket találhatunk, mint az eddigiek, illetve láthatjuk, hogy valamilyen eddig tanult módszer biztosan előnytelen lesz. Ez lesz például a B-fában tárolt adatok esete.

Adatbáziselemekből álló hierarchiák

A valóságban zárolhatunk teljes relációkat, de zárolhatjuk külön-külön ezek egyes blokkjait, sőt sorait is.

Adatbáziselemekből álló hierarchiák

A valóságban zárolhatunk teljes relációkat, de zárolhatjuk külön-külön ezek egyes blokkjait, sőt sorait is.
Minél nagyobb egységre rakunk zárat, annál könnyebb lesz a záradminisztráció, de annál több lesz a zárfeloldásra várás is és ezzel együtt a holtpont.

Adatbáziselemekből álló hierarchiák

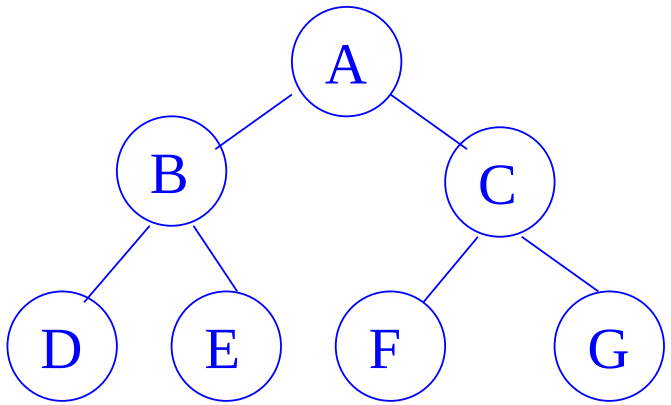
A valóságban zárolhatunk teljes relációkat, de zárolhatjuk külön-külön ezek egyes blokkjait, sőt sorait is.
Minél nagyobb egységre rakunk zárat, annál könnyebb lesz a záradminisztráció, de annál több lesz a zárfeloldásra várás is és ezzel együtt a holtpont.
Alkalmazástól függ, hogy mi éri meg jobban, de egy valami mindig közös: Elvárjuk azt, hogy ha az A adategység egy reláció, B pedig ennek egy blokkja, akkor az A-ra rakott zár zárolja B-t is, azaz pl. az egyszerű tranzakciómodellben ne lehessen $l_j(B)$-t kapni, ha $l_i(A)$ után még nem volt $u_i(A)$.

Adatbáziselemekből álló hierarchiák

A valóságban zárolhatunk teljes relációkat, de zárolhatjuk külön-külön ezek egyes blokkjait, sőt sorait is.
Minél nagyobb egységre rakunk zárat, annál könnyebb lesz a záradminisztráció, de annál több lesz a zárfeloldásra várás is és ezzel együtt a holtpont.
Alkalmazástól függ, hogy mi éri meg jobban, de egy valami mindig közös: Elvárjuk azt, hogy ha az A adategység egy reláció, B pedig ennek egy blokkja, akkor az A-ra rakott zár zárolja B-t is, azaz pl. az egyszerű tranzakciómodellben ne lehessen $l_j(B)$-t kapni, ha $l_i(A)$ után még nem volt $u_i(A)$. Ezt az eddigi technika még nem biztosítja, eddig ilyen összefüggéseket nem is vettünk figyelembe.

Adatbáziselemekből álló hierarchiák

A valóságban zárolhatunk teljes relációkat, de zárolhatjuk külön-külön ezek egyes blokkjait, sőt sorait is.
Minél nagyobb egységre rakunk zárat, annál könnyebb lesz a záradminisztráció, de annál több lesz a zárfeloldásra várás is és ezzel együtt a holtpont.
Alkalmazástól függ, hogy mi éri meg jobban, de egy valami mindig közös: Elvárjuk azt, hogy ha az A adategység egy reláció, B pedig ennek egy blokkja, akkor az A-ra rakott zár zárolja B-t is, azaz pl. az egyszerű tranzakciómodellben ne lehessen $l_j(B)$-t kapni, ha $l_i(A)$ után még nem volt $u_i(A)$. Ezt az eddigi technika még nem biztosítja, eddig ilyen összefüggéseket nem is vettünk figyelembe.
Egy ilyen lehetséges hierarchikus helyzet:



A: reláció

B,C: blokkjai

D,E,F,G: sorai

Figyelmeztető zármodell

Cél: olyan sorosítható ütemezés kikényszerítése, ami még az adategységek közötti hierarchiát is figyelembe veszi. Ez a hierarchia egy fával adott.

Figyelmeztető zármodell

Cél: olyan sorosítható ütemezés kikényszerítése, ami még az adategységek közötti hierarchiát is figyelembe veszi. Ez a hierarchia egy fával adott.
Egyszerű változat esetén háromféle zárművelet lesz: (lényegében az egyszerű tranzakciómodellnek felel meg):

Figyelmeztető zármodell

Cél: olyan sorosítható ütemezés kikényszerítése, ami még az adategységek közötti hierarchiát is figyelembe veszi. Ez a hierarchia egy fával adott.
Egyszerű változat esetén háromféle zárművelet lesz: (lényegében az egyszerű tranzakciómodellnek felel meg):

- $LOCK_i(A)$: T_i zárolja A-t (explicit lock) és minden leszármazottját (implicit lock), kizárólagosan, azaz ezek után más tranzakció se A-ra, se ennek leszármazottjára nem kaphat zárat.

Figyelmeztető zármodell

Cél: olyan sorosítható ütemezés kikényszerítése, ami még az adategységek közötti hierarchiát is figyelembe veszi. Ez a hierarchia egy fával adott.
Egyszerű változat esetén háromféle zárművelet lesz: (lényegében az egyszerű tranzakciómodellnek felel meg):

- $LOCK_i(A)$: T_i zárolja A-t (explicit lock) és minden leszármazottját (implicit lock), kizárólagosan, azaz ezek után más tranzakció se A-ra, se ennek leszármazottjára nem kaphat zárat.
- $WARN_i(A)$: T_i figyelmeztetést rak A-ra (gyerekeire nem), ez annak jelzésére szolgál, hogy T_i majd zárat akar kapni A valamely leszármazottjára

Figyelmeztető zármodell

Cél: olyan sorosítható ütemezés kikényszerítése, ami még az adategységek közötti hierarchiát is figyelembe veszi. Ez a hierarchia egy fával adott.
Egyszerű változat esetén háromféle zárművelet lesz: (lényegében az egyszerű tranzakciómodellnek felel meg):

- $LOCK_i(A)$: T_i zárolja A-t (explicit lock) és minden leszármazottját (implicit lock), kizárólagosan, azaz ezek után más tranzakció se A-ra, se ennek leszármazottjára nem kaphat zárat.
- $WARN_i(A)$: T_i figyelmeztetést rak A-ra (gyerekeire nem), ez annak jelzésére szolgál, hogy T_i majd zárat akar kapni A valamely leszármazottjára
- $UNLOCK_i(A)$: felszabadítja az A-ra rakott $LOCK$-ot és $WARN$-t, az implicit zár is lekerül A leszármazottjairól

A zárak használata

1. Az i-edik tranzakció, T_i, csak akkor olvashatja vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$ vagy $LOCK_i$ A valamelyik ősén) és ezt a zárat még azóta nem engedte fel.

A zárak használata

1. Az i-edik tranzakció, T_i, csak akkor olvashatja vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$ vagy $LOCK_i$ A valamelyik ősén) és ezt a zárat még azóta nem engedte fel.
2. $LOCK_i(A)$ és $WARN_i(A)$ után mindig van $UNLOCK_i(A)$.

A zárak használata

1. Az i-edik tranzakció, T_i, csak akkor olvashatja vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$ vagy $LOCK_i$ A valamelyik ősén) és ezt a zárat még azóta nem engedte fel.
2. $LOCK_i(A)$ és $WARN_i(A)$ után mindig van $UNLOCK_i(A)$.
3. Ha $LOCK_i$ van A-n, akkor se $WARN_j$ se $LOCK_j$ nem kerülhet már rá (ha $j \neq i$), de két különböző tranzakciónak lehet $WARN$-ja ugyanott. Vagyis a kompatibilitási mátrix:

	LOCK	WARN
LOCK	N	N
WARN	N	I

A figyelmeztető protokoll

A T_i tranzakció követi a figyelmeztető protokollt, ha zárkérései a fentieken kívül még a következőket is tudják:

(a) T_i első zárkérése $WARN_i$ vagy $LOCK_i$ a gyökérre

A figyelmeztető protokoll

A T_i tranzakció követi a figyelmeztető protokollt, ha zárkérései a fentieken kívül még a következőket is tudják:

(a) T_i első zárkérése $WARN_i$ vagy $LOCK_i$ a gyökérre

(b) Ezután $LOCK_i$ vagy $WARN_i$ csak akkor kérhető egy adategységre, ha $WARN_i$ már van az apján.

A figyelmeztető protokoll

A T_i tranzakció követi a figyelmeztető protokollt, ha zárkérései a fentieken kívül még a következőket is tudják:

(a) T_i első zárkérése $WARN_i$ vagy $LOCK_i$ a gyökérre

(b) Ezután $LOCK_i$ vagy $WARN_i$ csak akkor kérhető egy adategységre, ha $WARN_i$ már van az apján.

(c) $UNLOCK_i$ csak akkor kérhető egy adategységre, ha már nincs sem explicit $LOCK_i$, sem $WARN_i$ az adategység leszármazottjain

A figyelmeztető protokoll

A T_i tranzakció követi a figyelmeztető protokollt, ha zárkérései a fentieken kívül még a következőket is tudják:

(a) T_i első zárkérése $WARN_i$ vagy $LOCK_i$ a gyökérre

(b) Ezután $LOCK_i$ vagy $WARN_i$ csak akkor kérhető egy adategységre, ha $WARN_i$ már van az apján.

(c) $UNLOCK_i$ csak akkor kérhető egy adategységre, ha már nincs sem explicit $LOCK_i$, sem $WARN_i$ az adategység leszármazottjain

(d) Kétfázisú zárkérés van: $UNLOCK_i$ után nincs se $LOCK_i$, se $WARN_i$.

A figyelmeztető protokoll

A T_i tranzakció követi a figyelmeztető protokollt, ha zárkérései a fentieken kívül még a következőket is tudják:

(a) T_i első zárkérése $WARN_i$ vagy $LOCK_i$ a gyökérre

(b) Ezután $LOCK_i$ vagy $WARN_i$ csak akkor kérhető egy adategységre, ha $WARN_i$ már van az apján.

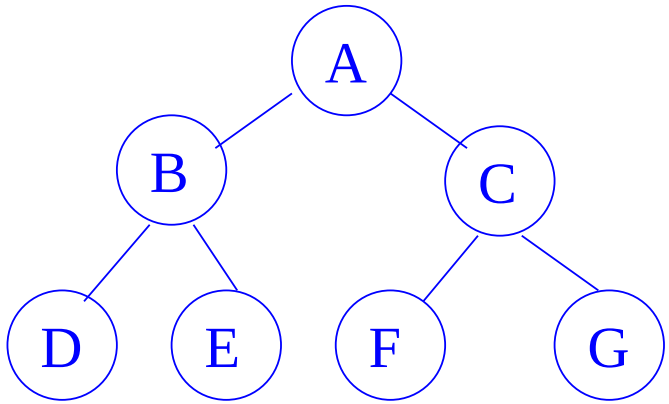
(c) $UNLOCK_i$ csak akkor kérhető egy adategységre, ha már nincs sem explicit $LOCK_i$, sem $WARN_i$ az adategység leszármazottjain

(d) Kétfázisú zárkérés van: $UNLOCK_i$ után nincs se $LOCK_i$, se $WARN_i$.

Az (a) és (b) pontok miatt a zárkérések felülről lefelé kúsznak a fában, a zárelengedések pedig a (c) miatt alulról felfele mennek az egyes tranzakciók esetén.

Példa

Az adategységek hierarchiája legyen (mint előbb):



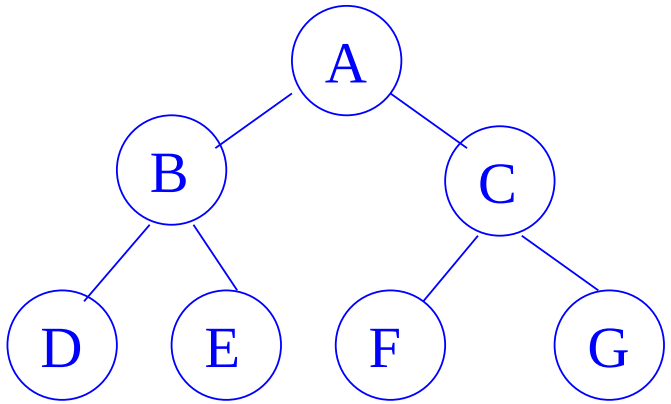
A: reláció

B,C: blokkjai

D,E,F,G: sorai

Példa

Az adategységek hierarchiája legyen (mint előbb):



A: reláció

B,C: blokkjai

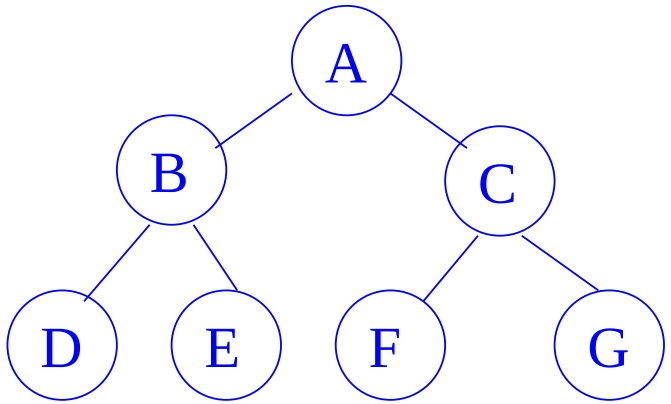
D,E,F,G: sorai

Az alábbi zárkérésekből és zárelengedésekből álló sorozat legális és mindhárom tranzakció követi a figyelmeztető protokollt.

$WARN_1(A)$, $WARN_2(A)$, $WARN_3(A)$, $WARN_1(B)$, $LOCK_2(C)$, $LOCK_1(D)$, $UNLOCK_2(C)$, $UNLOCK_1(D)$, $UNLOCK_2(A)$, $UNLOCK_1(B)$, $LOCK_3(B)$, $WARN_3(C)$, $LOCK_3(F)$, $UNLOCK_1(A)$, $UNLOCK_3(B)$, $UNLOCK_3(F)$, $UNLOCK_3(C)$, $UNLOCK_3(A)$

Példa

Az adategységek hierarchiája legyen (mint előbb):



A: reláció

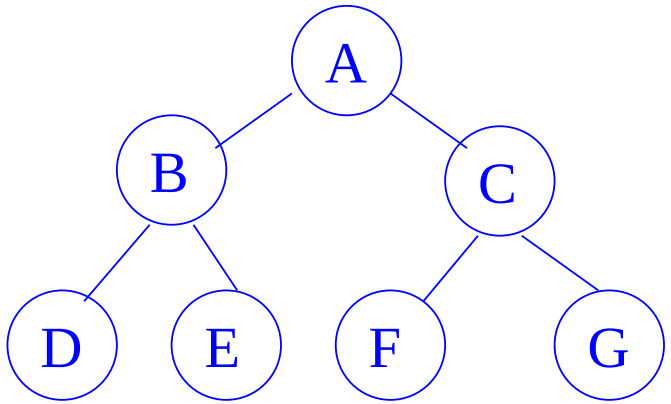
B,C: blokkjai

D,E,F,G: sorai

Az alábbi zárkérésekből és zárelengedésekből álló sorozat legális és mindhárom tranzakció követi a figyelmeztető protokollt.
$WARN_1(A)$, $WARN_2(A)$, $WARN_3(A)$, $WARN_1(B)$, $LOCK_2(C)$, $LOCK_1(D)$, $UNLOCK_2(C)$, $UNLOCK_1(D)$, $UNLOCK_2(A)$, $UNLOCK_1(B)$, $LOCK_3(B)$, $WARN_3(C)$, $LOCK_3(F)$, $UNLOCK_1(A)$, $UNLOCK_3(B)$, $UNLOCK_3(F)$, $UNLOCK_3(C)$, $UNLOCK_3(A)$
Azért legális, mert minden LOCK és WARN fel van engedve később és egyszerre csak több WARN van ugyanott, más nem.

Példa

Az adategységek hierarchiája legyen (mint előbb):



A: reláció

B,C: blokkjai

D,E,F,G: sorai

Az alábbi zárkérésekből és zárelengedésekből álló sorozat legális és mindhárom tranzakció követi a figyelmeztető protokollt.
$WARN_1(A)$, $WARN_2(A)$, $WARN_3(A)$, $WARN_1(B)$, $LOCK_2(C)$, $LOCK_1(D)$, $UNLOCK_2(C)$, $UNLOCK_1(D)$, $UNLOCK_2(A)$, $UNLOCK_1(B)$, $LOCK_3(B)$, $WARN_3(C)$, $LOCK_3(F)$, $UNLOCK_1(A)$, $UNLOCK_3(B)$, $UNLOCK_3(F)$, $UNLOCK_3(C)$, $UNLOCK_3(A)$
Azért legális, mert minden LOCK és WARN fel van engedve később és egyszerre csak több WARN van ugyanott, más nem.
A tranzakciók zárkérései pedig egyrészt 2PL szerint mennek, másrészt a zárkérések a fában felülről lefele mennek, a zárelengedések pedig alulról felfele.

Figyelmeztető protokoll II.

A figyelmeztető protokollra igaz az alábbi tétel:

Tétel

Ha a figyelmeztető zármodellben, egy legális ütemezésben minden tranzakció követi a figyelmeztető protokollt, akkor az ütemezés sorosítható és soha nem lesz egyszerre két különböző tranzakciónak zárja (se explicit, se implicit) ugyanazon az adategységen.

Figyelmeztető protokoll II.

A figyelmeztető protokollra igaz az alábbi tétel:

Tétel

Ha a figyelmeztető zármodellben, egy legális ütemezésben minden tranzakció követi a figyelmeztető protokollt, akkor az ütemezés sorosítható és soha nem lesz egyszerre két különböző tranzakciónak zárja (se explicit, se implicit) ugyanazon az adategységen.

Bizonyítás.

A sorosíthatóságot a kétfázisúság biztosítja (pontosabban nem bizonyítjuk).

Figyelmeztető protokoll II.

A figyelmeztető protokollra igaz az alábbi tétel:

Tétel

Ha a figyelmeztető zármodellben, egy legális ütemezésben minden tranzakció követi a figyelmeztető protokollt, akkor az ütemezés sorosítható és soha nem lesz egyszerre két különböző tranzakciónak zárja (se explicit, se implicit) ugyanazon az adategységen.

Bizonyítás.

A sorosíthatóságot a kétfázisúság biztosítja (pontosabban nem bizonyítjuk).
Zárkonfliktus pedig azért nem lesz, mert

- Két különböző tranzakciónak explicit LOCK-ja nem lehet soha ugyanott, ha az ütemezés legális.

Figyelmeztető protokoll II.

A figyelmeztető protokollra igaz az alábbi tétel:

Tétel

Ha a figyelmeztető zármodellben, egy legális ütemezésben minden tranzakció követi a figyelmeztető protokollt, akkor az ütemezés sorosítható és soha nem lesz egyszerre két különböző tranzakciónak zárja (se explicit, se implicit) ugyanazon az adategységen.

Bizonyítás.

A sorosíthatóságot a kétfázisúság biztosítja (pontosabban nem bizonyítjuk).
Zárkonfliktus pedig azért nem lesz, mert

- Két különböző tranzakciónak explicit LOCK-ja nem lehet soha ugyanott, ha az ütemezés legális.
- Egy explicit és egy implicit LOCK (két különböző tranzakciótól) nem lehet ugyanott: nem lehet, hogy egy A adatelemen $LOCK_i$ van és ezzel egyidejűleg egy leszármazottján (akin így implicit T_i lock van) van $LOCK_j$ is, mert ekkor A-n $WARN_j$-nek is kell lennie, de az nem lehet egy legális ütemezésben.

Figyelmeztető protokoll II.

A figyelmeztető protokollra igaz az alábbi tétel:

Tétel

Ha a figyelmeztető zármodellben, egy legális ütemezésben minden tranzakció követi a figyelmeztető protokollt, akkor az ütemezés sorosítható és soha nem lesz egyszerre két különböző tranzakciónak zárja (se explicit, se implicit) ugyanazon az adategységen.

Bizonyítás.

A sorosíthatóságot a kétfázisúság biztosítja (pontosabban nem bizonyítjuk).
Zárkonfliktus pedig azért nem lesz, mert

- Két különböző tranzakciónak explicit LOCK-ja nem lehet soha ugyanott, ha az ütemezés legális.
- Egy explicit és egy implicit LOCK (két különböző tranzakciótól) nem lehet ugyanott: nem lehet, hogy egy A adatelemen $LOCK_i$ van és ezzel egyidejűleg egy leszármazottján (akin így implicit T_i lock van) van $LOCK_j$ is, mert ekkor A-n $WARN_j$-nek is kell lennie, de az nem lehet egy legális ütemezésben.
- Két különböző tranzakciónak implicit LOCK-ja sem lehet ugyanazon a C adategységen, mert ekkor C két különböző felmenőjén a két különböző tranzakció két LOCK-jának kellene lennie, ami az előző pont értelmében nem lehet.

□

Figyelmeztető protokoll II.

Megjegyzés: Lehetne bonyolultabb zármód esetén is nézni figyelmeztető zárolást és figyelmeztető protokollt (az RLOCK/WLOCK modelnek megfelelően): lenne külön írási és olvasási figyelmeztetés is.

B-fában tárolt adategységek

Tekintsük most azt a helyzetet, amikor a zárolható adategységek egy fa csúcsaiban helyezkednek el, de a fa most nem az adategységek egymásba ágyazottságát mutatja (az adategységek most diszjunktak), hanem azt, hogy hogyan lehet elérni az adatokat.

B-fában tárolt adategységek

Tekintsük most azt a helyzetet, amikor a zárolható adategységek egy fa csúcsaiban helyezkednek el, de a fa most nem az adategységek egymásba ágyazottságát mutatja (az adategységek most diszjunktak), hanem azt, hogy hogyan lehet elérni az adatokat. Például a B-fa esetén a levelekhez csak úgy juthatunk el, ha a gyökértől indulva végigjárunk egy lefele vezető utat. Ahhoz, hogy beolvashassuk azt a levelet, ami nekünk kell, előtte be kell olvasnunk az összes felmenőjét (és ha csúcsvágás vagy csúcsösszevonás úgy kívánja, írnunk is kell őket).

B-fában tárolt adategységek

Tekintsük most azt a helyzetet, amikor a zárolható adategységek egy fa csúcsaiban helyezkednek el, de a fa most nem az adategységek egymásba ágyazottságát mutatja (az adategységek most diszjunktak), hanem azt, hogy hogyan lehet elérni az adatokat. Például a B-fa esetén a levelekhez csak úgy juthatunk el, ha a gyökértől indulva végigjárunk egy lefele vezető utat. Ahhoz, hogy beolvashassuk azt a levelet, ami nekünk kell, előtte be kell olvasnunk az összes felmenőjét (és ha csúcsvágás vagy csúcsösszevonás úgy kívánja, írnunk is kell őket).
Ilyenkor a szokásos technikák mennek ugyan, de nagyon előnytelenek lehetnek. Például a 2PL esetén egész addig kell tartani a zárat a gyökéren, amíg le nem értünk a levélhez, ami indokolatlanul sok várakozáshoz vezet.

B-fában tárolt adategységek

Tekintsük most azt a helyzetet, amikor a zárolható adategységek egy fa csúcsaiban helyezkednek el, de a fa most nem az adategységek egymásba ágyazottságát mutatja (az adategységek most diszjunktak), hanem azt, hogy hogyan lehet elérni az adatokat. Például a B-fa esetén a levelekhez csak úgy juthatunk el, ha a gyökértől indulva végigjárunk egy lefele vezető utat. Ahhoz, hogy beolvashassuk azt a levelet, ami nekünk kell, előtte be kell olvasnunk az összes felmenőjét (és ha csúcsvágás vagy csúcsösszevonás úgy kívánja, írnunk is kell őket).
Ilyenkor a szokásos technikák mennek ugyan, de nagyon előnytelenek lehetnek. Például a 2PL esetén egész addig kell tartani a zárat a gyökéren, amíg le nem értünk a levélhez, ami indokolatlanul sok várakozáshoz vezet.
Kéne másik módszer, ami ebben a speciális esetben biztosítja a sorosíthatóságot, de nem olyan szigorú, mint a 2PL. Ez lesz a faprotokoll.

Faprotokoll

Egyszerű tranzakciómodellben vagyunk (de ezt is lehetne RLOCK/WLOCK modellre kibővíteni), azaz

- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott

Faprotokoll

Egyszerű tranzakciómodellben vagyunk (de ezt is lehetne RLOCK/WLOCK modellre kibővíteni), azaz

- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott

A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.

Faprotokoll

Egyszerű tranzakciómodellben vagyunk (de ezt is lehetne RLOCK/WLOCK modellre kibővíteni), azaz

- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott

A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.
2. A későbbiekben azonban csak akkor kaphat zárat A-n, ha ekkor zárja van A apján.

Faprotokoll

Egyszerű tranzakciómodellben vagyunk (de ezt is lehetne RLOCK/WLOCK modellre kibővíteni), azaz

- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott

A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.
2. A későbbiekben azonban csak akkor kaphat zárat A-n, ha ekkor zárja van A apján.
3. Zárat bármikor fel lehet oldani (nem 2PL).

Faprotokoll

Egyszerű tranzakciómodellben vagyunk (de ezt is lehetne RLOCK/WLOCK modellre kibővíteni), azaz

- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott

A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.
2. A későbbiekben azonban csak akkor kaphat zárat A-n, ha ekkor zárja van A apján.
3. Zárat bármikor fel lehet oldani (nem 2PL).
4. Nem lehet újrazárolni, azaz ha T_i elengedte egy A adategység zárját, akkor később nem kérhet rá újra (még akkor sem, ha A apján még megvan a zárja).

Faprotokoll

Egyszerű tranzakciómodellben vagyunk (de ezt is lehetne RLOCK/WLOCK modellre kibővíteni), azaz

- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott

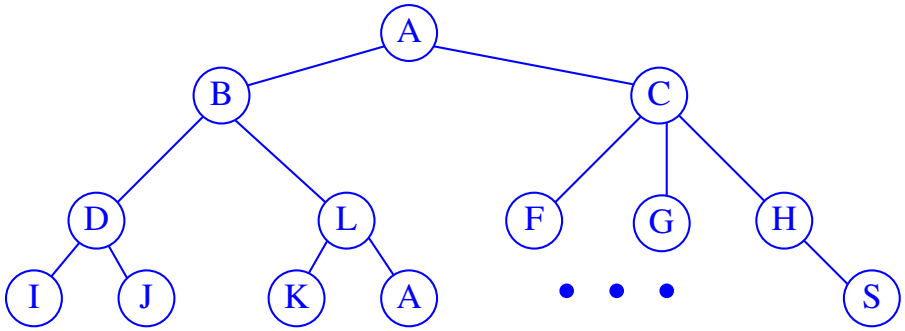
A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.
2. A későbbiekben azonban csak akkor kaphat zárat A-n, ha ekkor zárja van A apján.
3. Zárat bármikor fel lehet oldani (nem 2PL).
4. Nem lehet újrazárolni, azaz ha T_i elengedte egy A adategység zárját, akkor később nem kérhet rá újra (még akkor sem, ha A apján még megvan a zárja).

Tétel

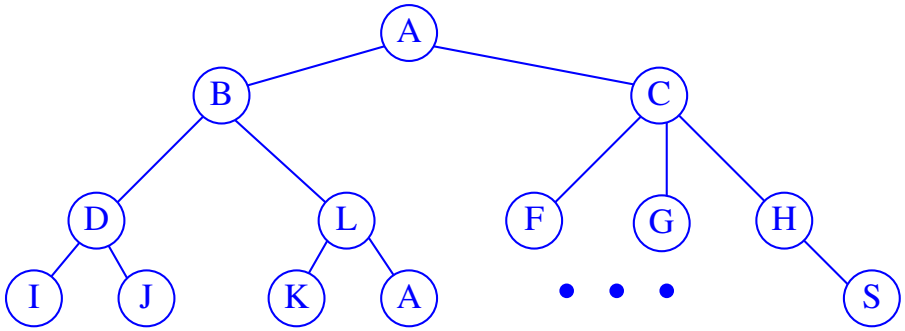
(Bizonyítás nélkül) Ha minden tranzakció követi a faprotokollt egy legális ütemezésben, akkor az ütemezés sorosítható lesz, noha nem feltétlenül lesz 2PL.

Példa



Tekintsük ezt a B_3-fát. Az A-tól H-ig levő adategységek a fa belső csúcsai, itt mutatók és keresést segítő kulcsok vannak, a levelekben (I-től S-ig) pedig a keresési kulcs szerint rendezetten vannak a tárolt adatok, amik között keresünk. Most feltesszük, hogy egy levélben egy tárolt elem van.

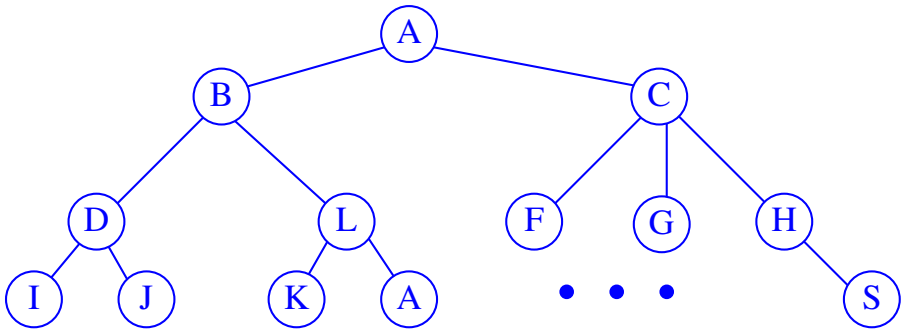
Példa



Tekintsük ezt a B_3-fát. Az A-tól H-ig levő adategységek a fa belső csúcsai, itt mutatók és keresést segítő kulcsok vannak, a levelekben (I-től S-ig) pedig a keresési kulcs szerint rendezetten vannak a tárolt adatok, amik között keresünk. Most feltesszük, hogy egy levélben egy tárolt elem van.

Ha mondjuk az *I*-ben, *J*-ben és *K*-ban tárolt elemek keresési kulcsa 1, 3 és 10, és be akarunk szúrni egy olyan elemet, ahol a kulcs értéke 4, akkor először olvasni kell *A*-t, *B*-t és *D*-t, majd írni is kell *D*-t.

Példa



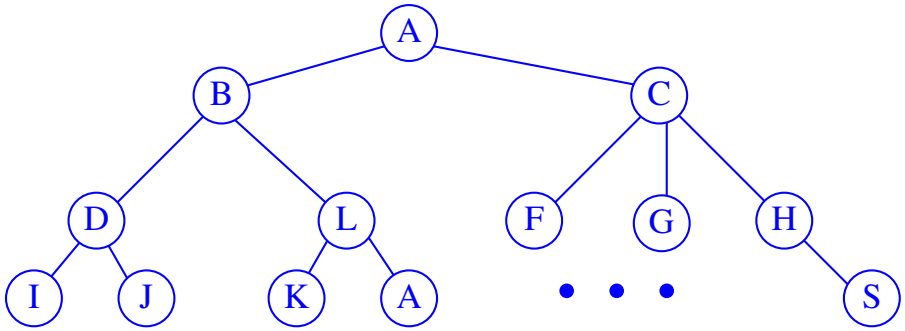
Tekintsük ezt a B_3-fát. Az A-tól H-ig levő adategységek a fa belső csúcsai, itt mutatók és keresést segítő kulcsok vannak, a levelekben (I-től S-ig) pedig a keresési kulcs szerint rendezetten vannak a tárolt adatok, amik között keresünk. Most feltesszük, hogy egy levélben egy tárolt elem van.

Ha mondjuk az *I*-ben, *J*-ben és *K*-ban tárolt elemek keresési kulcsa 1, 3 és 10, és be akarunk szúrni egy olyan elemet, ahol a kulcs értéke 4, akkor először olvasni kell *A*-t, *B*-t és *D*-t, majd írni is kell *D*-t.

Ekkor a megfelelő (faprotokoll szerinti, legális) ütemezés eleje

$LOCK_i(A)$, $LOCK_i(B)$, $UNLOCK_i(A)$ mert *B* beolvasása után látjuk, hogy neki csak két gyereke van, ha kell is csúcsvágás, az *A*-t biztos nem érinti, *A*-t nem kell majd írni. Csak addig kellett fogni *A*-n a zárat, amíg *B*-re is megkaptuk.

Példa



Tekintsük ezt a B_3-fát. Az A-tól H-ig levő adategységek a fa belső csúcsai, itt mutatók és keresést segítő kulcsok vannak, a levelekben (I-től S-ig) pedig a keresési kulcs szerint rendezetten vannak a tárolt adatok, amik között keresünk. Most feltesszük, hogy egy levélben egy tárolt elem van.

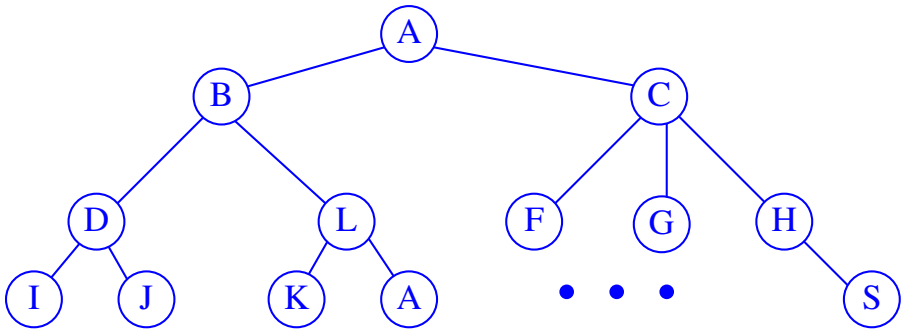
Ha mondjuk az *I*-ben, *J*-ben és *K*-ban tárolt elemek keresési kulcsa 1, 3 és 10, és be akarunk szúrni egy olyan elemet, ahol a kulcs értéke 4, akkor először olvasni kell *A*-t, *B*-t és *D*-t, majd írni is kell *D*-t.

Ekkor a megfelelő (faprotokoll szerinti, legális) ütemezés eleje

$LOCK_i(A)$, $LOCK_i(B)$, $UNLOCK_i(A)$ mert *B* beolvasása után látjuk, hogy neki csak két gyereke van, ha kell is csúcsvágás, az *A*-t biztos nem érinti, *A*-t nem kell majd írni. Csak addig kellett fogni *A*-n a zárat, amíg *B*-re is megkaptuk.

Ezután $LOCK_i(D)$, $UNLOCK_i(B)$, mert látjuk, hogy *D*-nek csak két gyereke van, ezért *B*-t biztos nem kell írni.

Példa



Tekintsük ezt a B_3-fát. Az A-tól H-ig levő adategységek a fa belső csúcsai, itt mutatók és keresést segítő kulcsok vannak, a levelekben (I-től S-ig) pedig a keresési kulcs szerint rendezetten vannak a tárolt adatok, amik között keresünk. Most feltesszük, hogy egy levélben egy tárolt elem van.

Ha mondjuk az I-ben, J-ben és K-ban tárolt elemek keresési kulcsa $1, 3$ és 10, és be akarunk szúrni egy olyan elemet, ahol a kulcs értéke 4, akkor először olvasni kell A-t, B-t és D-t, majd írni is kell D-t.

Ekkor a megfelelő (faprotokoll szerinti, legális) ütemezés eleje

$LOCK_i(A)$, $LOCK_i(B)$, $UNLOCK_i(A)$ mert B beolvasása után látjuk, hogy neki csak két gyereke van, ha kell is csúcsvágás, az A-t biztos nem érinti, A-t nem kell majd írni. Csak addig kellett fogni A-n a zárat, amíg B-re is megkaptuk.

Ezután $LOCK_i(D)$, $UNLOCK_i(B)$, mert látjuk, hogy D-nek csak két gyereke van, ezért B-t biztos nem kell írni.

Innen tovább: $UNLOCK_i(D)$, amikor már megtörtént az új levél beszúrása és D-ben is beállítottuk a mutatókat.

Példa II.

Tanulság:

- Faprotokoll szerint ment az ütemezés $\Longrightarrow$ jó lesz
- Nem 2PL és ezzel nyertünk is sokat, mert amint megvolt $UNLOCK_i(A)$, akkor rögtön indulhat a következő beszúrás, ha az a fa jobb oldali ágán fut le. Ha 2PL lett volna, akkor $LOCK_i(D)$-ig kellene várni ezzel.

Adatbázisok elmélete

Sorosíthatóság időbélyegekkel

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

24. előadás

Sorosíthatóság időbélyegekkel

Eddig a zárakat vizsgáltuk, mint egy lehetséges technikát a sorosíthatóság kikényszerítésére. Másik lehetőség: **időbélyeges tranzakciókezelés**.

Sorosíthatóság időbélyegekkel

Eddig a zárakat vizsgáltuk, mint egy lehetséges technikát a sorosíthatóság kikényszerítésére. Másik lehetőség: **időbélyeges tranzakciókezelés**.
Ez optimistább, illetve agresszívabb, mint a zárak használata: hagyja a tranzakciókat szabadon futni (ellentétben a záraknál látott protokollokkal), de ha baj lenne, akkor agresszívan közbelép (ABORT).

Sorosíthatóság időbélyegekkel

Eddig a zárakat vizsgáltuk, mint egy lehetséges technikát a sorosíthatóság kikényszerítésére. Másik lehetőség: **időbélyeges tranzakciókezelés**.
Ez optimistább, illetve agresszívabb, mint a zárak használata: hagyja a tranzakciókat szabadon futni (ellentétben a záraknál látott protokollokkal), de ha baj lenne, akkor agresszívan közbelép (ABORT).
Akkor jó, ha ritkán lesz ABORT, ha valószínűleg kevés lesz a sorosítási probléma.

Sorosíthatóság időbélyegekkel

Eddig a zárakat vizsgáltuk, mint egy lehetséges technikát a sorosíthatóság kikényszerítésére. Másik lehetőség: **időbélyeges tranzakciókezelés**.
Ez optimistább, illetve agresszívabb, mint a zárak használata: hagyja a tranzakciókat szabadon futni (ellentétben a záraknál látott protokollokkal), de ha baj lenne, akkor agresszívan közbelép (ABORT).
Akkor jó, ha ritkán lesz ABORT, ha valószínűleg kevés lesz a sorosítási probléma.
Fő elv: minden tranzakciónak van egy időbélyege: $t(T_i)$ a T_i tranzakcióé. Az időbélyegek egyediek, növő sorrendben adja ki őket az ütemező, ahogy indulnak a tranzakciók.

Sorosíthatóság időbélyegekkel

Eddig a zárakat vizsgáltuk, mint egy lehetséges technikát a sorosíthatóság kikényszerítésére. Másik lehetőség: **időbélyeges tranzakciókezelés**.
Ez optimistább, illetve agresszívabb, mint a zárak használata: hagyja a tranzakciókat szabadon futni (ellentétben a záraknál látott protokollokkal), de ha baj lenne, akkor agresszívan közbelép (ABORT).
Akkor jó, ha ritkán lesz ABORT, ha valószínűleg kevés lesz a sorosítási probléma.
Fő elv: minden tranzakciónak van egy időbélyege: $t(T_i)$ a T_i tranzakcióé. Az időbélyegek egyediek, növő sorrendben adja ki őket az ütemező, ahogy indulnak a tranzakciók.
Az ütemező célja: az időbélyegek növő sorrendjéhez tartozó soros ütemezéssel azonos hatású ütemezést enged csak lefutni, minden olyan kérést letilt (és a megfelelő tranzakciót ABORT-álja), ami ez ellen tesz.

Sorosíthatóság időbélyegekkel

Eddig a zárakat vizsgáltuk, mint egy lehetséges technikát a sorosíthatóság kikényszerítésére. Másik lehetőség: **időbélyeges tranzakciókezelés**.
Ez optimistább, illetve agresszívabb, mint a zárak használata: hagyja a tranzakciókat szabadon futni (ellentétben a záraknál látott protokollokkal), de ha baj lenne, akkor agresszívan közbelép (ABORT).
Akkor jó, ha ritkán lesz ABORT, ha valószínűleg kevés lesz a sorosítási probléma.
Fő elv: minden tranzakciónak van egy időbélyege: $t(T_i)$ a T_i tranzakcióé. Az időbélyegek egyediek, növő sorrendben adja ki őket az ütemező, ahogy indulnak a tranzakciók.
Az ütemező célja: az időbélyegek növő sorrendjéhez tartozó soros ütemezéssel azonos hatású ütemezést enged csak lefutni, minden olyan kérést letilt (és a megfelelő tranzakciót ABORT-álja), ami ez ellen tesz.
Például, ha $t(T_1) = 120$, $t(T_2) = 90$ és $t(T_3) = 130$, akkor a cél a $T_2 T_1 T_3$ soros sorrenddel azonos hatású ütemezés.

Az időbélyeges tranzakciókezelés szabályai

Megkülönböztetünk írás és olvasás műveletet, továbbá minden A adatelemhez hozzárendelünk egy olvasási és egy írási időt $(r(A), w(A))$, melyek jelentése:

- r(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i olvasta már A-t

Az időbélyeges tranzakciókezelés szabályai

Megkülönböztetünk írás és olvasás műveletet, továbbá minden A adatelemhez hozzárendelünk egy olvasási és egy írási időt $(r(A), w(A))$, melyek jelentése:

- r(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i olvasta már A-t
- w(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i írta már A-t

Az időbélyeges tranzakciókezelés szabályai

Megkülönböztetünk írás és olvasás műveletet, továbbá minden A adatelemhez hozzárendelünk egy olvasási és egy írási időt $(r(A), w(A))$, melyek jelentése:

- r(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i olvasta már A-t
- w(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i írta már A-t

Az ütemező mit csinál, hogy kikényszerítse az időbélyegek szerinti növő soros ütemezés hatását?

Az időbélyeges tranzakciókezelés szabályai

Megkülönböztetünk írás és olvasás műveletet, továbbá minden A adatelemhez hozzárendelünk egy olvasási és egy írási időt $(r(A), w(A))$, melyek jelentése:

- r(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i olvasta már A-t
- w(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i írta már A-t

Az ütemező mit csinál, hogy kikényszerítse az időbélyegek szerinti növő soros ütemezés hatását?

- minden induló tranzakciónak legenerál egy időbélyeget, egyedit, növően, ez lesz a tranzakció egész futása alatt az ő időbélyege

Az időbélyeges tranzakciókezelés szabályai

Megkülönböztetünk írás és olvasás műveletet, továbbá minden A adatelemhez hozzárendelünk egy olvasási és egy írási időt $(r(A), w(A))$, melyek jelentése:

- r(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i olvasta már A-t
- w(A)= a legnagyobb olyan $t(T_i)$, amire igaz, ahogy T_i írta már A-t

Az ütemező mit csinál, hogy kikényszerítse az időbélyegek szerinti növő soros ütemezés hatását?

- minden induló tranzakciónak legenerál egy időbélyeget, egyedit, növően, ez lesz a tranzakció egész futása alatt az ő időbélyege
- ha a T tranzakció bármit csinálni szeretne egy A adategységgel, akkor mielőtt ezt megengedné, megvizsgálja $t(T)$, és $r(A)$ illetve $w(A)$ kapcsolatát és a következőképpen cselekszik.

Szabályok

1. Ha T olvasná A-t, de $t(T) < w(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy írja)

Szabályok

1. Ha T olvasná A-t, de $t(T) < w(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy írja)
2. Ha T írná A-t, de $t(T) < r(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy olvassa)

Szabályok

1. Ha T olvasná A-t, de $t(T) < w(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy írja)
2. Ha T írná A-t, de $t(T) < r(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy olvassa)
3. Ha T olvasná A-t, $t(T) \geq w(A)$, de $t(T) < r(A)$, akkor T olvashatja A-t és $r(A)$ marad, ami volt és persze $w(A)$ is (mert ugyan egy nagyobb időbélyegű tranzakciónak már megengedtük, hogy olvassa A-t, de ez nem baj, ettől még kijöhet a kívánt soros ütemezés hatása)

Szabályok

1. Ha T olvasná A-t, de $t(T) < w(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy írja)
2. Ha T írná A-t, de $t(T) < r(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy olvassa)
3. Ha T olvasná A-t, $t(T) \geq w(A)$, de $t(T) < r(A)$, akkor T olvashatja A-t és $r(A)$ marad, ami volt és persze $w(A)$ is (mert ugyan egy nagyobb időbélyegű tranzakciónak már megengedtük, hogy olvassa A-t, de ez nem baj, ettől még kijöhet a kívánt soros ütemezés hatása)
4. Ha T írná A-t, $t(T) \geq r(A)$, de $t(T) < w(A)$, akkor nem történik meg az írás, de nem is lesz ABORT T se és $r(A)$ és $w(A)$ marad, ami volt (mivel egy nagyobb időbélyegű tranzakciónak már megengedtük, hogy írja A-t, ezért a kívánt soros hatásban úgyse látszódik ez az írás)

Szabályok

1. Ha T olvasná A-t, de $t(T) < w(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy írja)
2. Ha T írná A-t, de $t(T) < r(A)$, akkor ABORT T (mert egy nagyobb időbélyegű, azaz T után következő tranzakciónak már megengedte, hogy olvassa)
3. Ha T olvasná A-t, $t(T) \geq w(A)$, de $t(T) < r(A)$, akkor T olvashatja A-t és $r(A)$ marad, ami volt és persze $w(A)$ is (mert ugyan egy nagyobb időbélyegű tranzakciónak már megengedtük, hogy olvassa A-t, de ez nem baj, ettől még kijöhet a kívánt soros ütemezés hatása)
4. Ha T írná A-t, $t(T) \geq r(A)$, de $t(T) < w(A)$, akkor nem történik meg az írás, de nem is lesz ABORT T se és $r(A)$ és $w(A)$ marad, ami volt (mivel egy nagyobb időbélyegű tranzakciónak már megengedtük, hogy írja A-t, ezért a kívánt soros hatásban úgyse látszódik ez az írás)
5. Ha T olvasná vagy írná A-t, és $t(T) \geq w(A)$ és $t(T) \geq r(A)$, akkor engedjük és $r(A)$ illetve $w(A)$ változik, attól függően, hogy írás vagy olvasás történt

Példa

Legyenek a tranzakciók időbélyegei $t(T_1) = 20$, $t(T_2) = 10$ (vagyis cél a $T_2 T_1$ hatása) és tekintsük az alábbi kéréssorozatot:
$READ_2(A)$, $READ_1(A)$, $WRITE_1(C)$, $WRITE_2(C)$, $WRITE_2(A)$

Példa

Legyenek a tranzakciók időbélyegei $t(T_1) = 20$, $t(T_2) = 10$ (vagyis cél a $T_2 T_1$ hatása) és tekintsük az alábbi kéréssorozatot:
$READ_2(A)$, $READ_1(A)$, $WRITE_1(C)$, $WRITE_2(C)$, $WRITE_2(A)$
Hogyan változnak az olvasási és írási idők (kezdetben nullák) és mit csinál az ütemező? Lesz-e ABORT?

Példa

Legyenek a tranzakciók időbélyegei $t(T_1) = 20$, $t(T_2) = 10$ (vagyis cél a $T_2 T_1$ hatása) és tekintsük az alábbi kéréssorozatot:
$READ_2(A)$, $READ_1(A)$, $WRITE_1(C)$, $WRITE_2(C)$, $WRITE_2(A)$
Hogyan változnak az olvasási és írási idők (kezdetben nullák) és mit csinál az ütemező? Lesz-e ABORT?

Kérés	r(A)	w(A)	r(C)	w(C)	Magyarázat
	0	0	0	0	kezdetben minden nulla
$READ_2(A)$	10	0	0	0	5. eset $\Longrightarrow$ mehet
$READ_1(A)$	20	0	0	0	5. eset $\Longrightarrow$ mehet
$WRITE_1(C)$	20	0	0	20	5. eset $\Longrightarrow$ mehet
$WRITE_2(C)$	20	0	0	20	4. eset $\Longrightarrow$ nem mehet, de nincs ABORT se
$WRITE_2(A)$	20	0	0	20	2. eset $\Longrightarrow$ ABORT T_2

Megjegyzések

1. A szabályok 4. pontjánál (ahol nem volt se ABORT, se írás) egyes források ABORT-ot rendelnek el. Ennek oka, hogy az általunk definált szabályok alkalmazása esetén előfordulhat a következő kellemetlen jelenség:
   Ha az a T_i tranzakció, aki beállította $w(A)$ értékét (aminél az írni akaró T tranzakció időbélyege kisebb) esetleg ABORT-ál és emiatt vissza kell csinálni T_i összes hatását, akkor T hatásának látszania kellene, de nem fog, pedig T lefutott hiba nélkül.

Megjegyzések

1. A szabályok 4. pontjánál (ahol nem volt se ABORT, se írás) egyes források ABORT-ot rendelnek el. Ennek oka, hogy az általunk definált szabályok alkalmazása esetén előfordulhat a következő kellemetlen jelenség:
   Ha az a T_i tranzakció, aki beállította $w(A)$ értékét (aminél az írni akaró T tranzakció időbélyege kisebb) esetleg ABORT-ál és emiatt vissza kell csinálni T_i összes hatását, akkor T hatásának látszania kellene, de nem fog, pedig T lefutott hiba nélkül. Ha a 4.pont esetén ABORT-ot rendelünk el, akkor ez a gond nincsen. Vannak azonban technikák, amikkel akkor is meggátolható ez a jelenség, ha úgy járunk el, ahogy megadtuk a 4. pontnál a tennivalókat (most nem nézzük, hogy mik ezek a technikák), ezért nem kell az ABORT ebben az esetben.

Megjegyzések

1. A szabályok 4. pontjánál (ahol nem volt se ABORT, se írás) egyes források ABORT-ot rendelnek el. Ennek oka, hogy az általunk definált szabályok alkalmazása esetén előfordulhat a következő kellemetlen jelenség:
   Ha az a T_i tranzakció, aki beállította $w(A)$ értékét (aminél az írni akaró T tranzakció időbélyege kisebb) esetleg ABORT-ál és emiatt vissza kell csinálni T_i összes hatását, akkor T hatásának látszania kellene, de nem fog, pedig T lefutott hiba nélkül. Ha a 4.pont esetén ABORT-ot rendelünk el, akkor ez a gond nincsen. Vannak azonban technikák, amikkel akkor is meggátolható ez a jelenség, ha úgy járunk el, ahogy megadtuk a 4. pontnál a tennivalókat (most nem nézzük, hogy mik ezek a technikák), ezért nem kell az ABORT ebben az esetben.
2. Az időbélyeges módszer a zárhasználat alternatívája. Az időbélyeges módszernél ha sok az ABORT, akkor sokat kell majd dolgoznunk a visszaállítással (ezért akkor javasolt, ha kevés a közös elemeken történő írás);

Megjegyzések

1. A szabályok 4. pontjánál (ahol nem volt se ABORT, se írás) egyes források ABORT-ot rendelnek el. Ennek oka, hogy az általunk definált szabályok alkalmazása esetén előfordulhat a következő kellemetlen jelenség:
   Ha az a T_i tranzakció, aki beállította $w(A)$ értékét (aminél az írni akaró T tranzakció időbélyege kisebb) esetleg ABORT-ál és emiatt vissza kell csinálni T_i összes hatását, akkor T hatásának látszania kellene, de nem fog, pedig T lefutott hiba nélkül. Ha a 4.pont esetén ABORT-ot rendelünk el, akkor ez a gond nincsen. Vannak azonban technikák, amikkel akkor is meggátolható ez a jelenség, ha úgy járunk el, ahogy megadtuk a 4. pontnál a tennivalókat (most nem nézzük, hogy mik ezek a technikák), ezért nem kell az ABORT ebben az esetben.
2. Az időbélyeges módszer a zárhasználat alternatívája. Az időbélyeges módszernél ha sok az ABORT, akkor sokat kell majd dolgoznunk a visszaállítással (ezért akkor javasolt, ha kevés a közös elemeken történő írás); a zárak hátránya pedig az, hogy karban kell tartani a zártáblát és a korlátozások miatt sok lehet a várakozás és a holtpont.

Megjegyzések

1. A szabályok 4. pontjánál (ahol nem volt se ABORT, se írás) egyes források ABORT-ot rendelnek el. Ennek oka, hogy az általunk definált szabályok alkalmazása esetén előfordulhat a következő kellemetlen jelenség:
   Ha az a T_i tranzakció, aki beállította $w(A)$ értékét (aminél az írni akaró T tranzakció időbélyege kisebb) esetleg ABORT-ál és emiatt vissza kell csinálni T_i összes hatását, akkor T hatásának látszania kellene, de nem fog, pedig T lefutott hiba nélkül. Ha a 4.pont esetén ABORT-ot rendelünk el, akkor ez a gond nincsen. Vannak azonban technikák, amikkel akkor is meggátolható ez a jelenség, ha úgy járunk el, ahogy megadtuk a 4. pontnál a tennivalókat (most nem nézzük, hogy mik ezek a technikák), ezért nem kell az ABORT ebben az esetben.
2. Az időbélyeges módszer a zárhasználat alternatívája. Az időbélyeges módszernél ha sok az ABORT, akkor sokat kell majd dolgoznunk a visszaállítással (ezért akkor javasolt, ha kevés a közös elemeken történő írás); a zárak hátránya pedig az, hogy karban kell tartani a zártáblát és a korlátozások miatt sok lehet a várakozás és a holtpont.
3. Vannak még más módszerek is a sorosíthatóság elérésére, pl. érvényesítés.

Időbélyegek↔ zárak

Egyik se jobb egyértelműen, mint a másik. Van, hogy mind a kettő ugyanazokat a kéréseket hagyja lefutni:

(a) Ha T_2 előbb indul, mint T_1, akkor a $READ_2(B)$, $READ_1(A)$, $WRITE_1(C)$, $WRITE_2(C)$ műveletsort egy időbélyegesen dolgozó ütemező nem hagyja lefutni, mert a T_2T_1 soros sorrenddel ez nem lesz azonos hatású. Viszont RLOCK/WLOCK zárolás esetén van olyan legális zárkérés, amit az ütemező sorosíthatónak fog találni.

Időbélyegek↔ zárak

Egyik se jobb egyértelműen, mint a másik. Van, hogy mind a kettő ugyanazokat a kéréseket hagyja lefutni:

(a) Ha T_2 előbb indul, mint T_1, akkor a $READ_2(B)$, $READ_1(A)$, $WRITE_1(C)$, $WRITE_2(C)$ műveletsort egy időbélyegesen dolgozó ütemező nem hagyja lefutni, mert a T_2T_1 soros sorrenddel ez nem lesz azonos hatású. Viszont RLOCK/WLOCK zárolás esetén van olyan legális zárkérés, amit az ütemező sorosíthatónak fog találni.

(b) A $READ_1(A)$, $WRITE_2(A)$, $WRITE_1(A)$, $WRITE_1(B)$, $WRITE_2(B)$, $WRITE_3(A)$ műveletsort (itt T_1 indul előbb), bárhogy is kérjük a zárakat legálisan, nem hagyja lefutni egy RLOCK/WLOCK zárakat használó ütemező (T_2 és T_1 között mindkét irányban lesz él a sorosítási gráfban), de időbélyeggel lefut ez a műveletsor.

Védekezés hibák ellen, helyreállítás

Alapprobléma: nem fut le valamelyik tranzakció (sérül az atomiság) és emiatt inkonzisztens lesz az adatbázis.

Védekezés hibák ellen, helyreállítás

Alapprobléma: nem fut le valamelyik tranzakció (sérül az atomiság) és emiatt inkonzisztens lesz az adatbázis.
Ennek okai lehetnek:

1. tranzakcióhiba, programhiba
2. ütemező által elrendelt ABORT (holtpont vagy sorosíthatóság miatt)
3. rendszerhiba: belső tár sérül
4. médiahiba: háttértár is sérül

Védekezés hibák ellen, helyreállítás

Alapprobléma: nem fut le valamelyik tranzakció (sérül az atomiság) és emiatt inkonzisztens lesz az adatbázis.
Ennek okai lehetnek:

1. tranzakcióhiba, programhiba
2. ütemező által elrendelt ABORT (holtpont vagy sorosíthatóság miatt)
3. rendszerhiba: belső tár sérül
4. médiahiba: háttértár is sérül

Cél mindegyik esetben az, hogy újra konzisztens állapotba hozzuk az adatbázist (visszacsinálás vagy befejezés) úgy, hogy a tartósság megmaradjon: ha egy tranzakció már befejezte a munkáját, akkor annak hatása ne vesszen el.

Védekezés hibák ellen, helyreállítás

Alapprobléma: nem fut le valamelyik tranzakció (sérül az atomiság) és emiatt inkonzisztens lesz az adatbázis.
Ennek okai lehetnek:

1. tranzakcióhiba, programhiba
2. ütemező által elrendelt ABORT (holtpont vagy sorosíthatóság miatt)
3. rendszerhiba: belső tár sérül
4. médiahiba: háttértár is sérül

Cél mindegyik esetben az, hogy újra konzisztens állapotba hozzuk az adatbázist (visszacsinálás vagy befejezés) úgy, hogy a tartósság megmaradjon: ha egy tranzakció már befejezte a munkáját, akkor annak hatása ne vesszen el.
Az utolsó fajta hibával nem foglalkozunk, erre a szokásos eljárások mennek (archiválás, duplikálás).

Alapfogalmak

Feltevés, hogy a végig lefutott tranzakciók konzisztens állapotból konzisztens állapotba viszik az adatbázist, ezért baj csak akkor lehet, ha félbemaradnak.

Alapfogalmak

Feltevés, hogy a végig lefutott tranzakciók konzisztens állapotból konzisztens állapotba viszik az adatbázist, ezért baj csak akkor lehet, ha félbemaradnak.
Fontos eszköz a hiba utáni helyreállításban:
COMMIT pont: az a pont, amikor a tranzakció minden érdemi munkával megvan, programhiba vagy ütemező miatt ABORT már biztos nem lehet.

Alapfogalmak

Feltevés, hogy a végig lefutott tranzakciók konzisztens állapotból konzisztens állapotba viszik az adatbázist, ezért baj csak akkor lehet, ha félbemaradnak.
Fontos eszköz a hiba utáni helyreállításban:
COMMIT pont: az a pont, amikor a tranzakció minden érdemi munkával megvan, programhiba vagy ütemező miatt ABORT már biztos nem lehet. Nem biztos, hogy ekkor minden hatása látszik is már a tranzakciónak, lehet, hogy nincs minden írása véglegesítve, de minden készen áll már erre.

Alapfogalmak

Feltevés, hogy a végig lefutott tranzakciók konzisztens állapotból konzisztens állapotba viszik az adatbázist, ezért baj csak akkor lehet, ha félbemaradnak.
Fontos eszköz a hiba utáni helyreállításban:
COMMIT pont: az a pont, amikor a tranzakció minden érdemi munkával megvan, programhiba vagy ütemező miatt ABORT már biztos nem lehet. Nem biztos, hogy ekkor minden hatása látszik is már a tranzakciónak, lehet, hogy nincs minden írása véglegesítve, de minden készen áll már erre.
Fontos fogalom még:
Piszkos adat: Olyan adat, amit még nem COMMIT-ált tranzakció (azaz olyan, ami még meghalhat) írt az adatbázisba.

Alapfogalmak

Feltevés, hogy a végig lefutott tranzakciók konzisztens állapotból konzisztens állapotba viszik az adatbázist, ezért baj csak akkor lehet, ha félbemaradnak.
Fontos eszköz a hiba utáni helyreállításban:
COMMIT pont: az a pont, amikor a tranzakció minden érdemi munkával megvan, programhiba vagy ütemező miatt ABORT már biztos nem lehet. Nem biztos, hogy ekkor minden hatása látszik is már a tranzakciónak, lehet, hogy nincs minden írása véglegesítve, de minden készen áll már erre.
Fontos fogalom még:
Piszkos adat: Olyan adat, amit még nem COMMIT-ált tranzakció (azaz olyan, ami még meghalhat) írt az adatbázisba. Ha ilyet olvas egy másik tranzakció, akkor baj lehet, ha az első ABORT-ál, de a második nem.

T_1	T_2
LOCK(A)	
READ(A)	
A:=A+100	
WRITE(A)	
LOCK(B)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	$A := A \cdot 25$
READ(B)	
	WRITE(A)
	COMMIT
	UNLOCK(A)
$B := \frac{B}{A}$	
$\Downarrow$	
ABORT	

Példa piszkos adatból eredő hibára zárolásos ütemezés esetén (persze időbélyegekkel is van ilyen):

Ha az osztáskor A értéke éppen 0, akkor T_1 ABORT-ál, és emiatt sok baj lesz:

T_1	T_2
LOCK(A)	
READ(A)	
A:=A+100	
WRITE(A)	
LOCK(B)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	$A := A \cdot 25$
READ(B)	
	WRITE(A)
	COMMIT
	UNLOCK(A)
$B := \frac{B}{A}$	
$\Downarrow$	
ABORT	

Példa piszkos adatból eredő hibára zárolásos ütemezés esetén (persze időbélyegekkel is van ilyen):

Ha az osztáskor A értéke éppen 0, akkor T_1 ABORT-ál, és emiatt sok baj lesz:

- B-n zár marad, ezt fel kell oldani

T_1	T_2
LOCK(A)	
READ(A)	
A:=A+100	
WRITE(A)	
LOCK(B)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	$A := A \cdot 25$
READ(B)	
	WRITE(A)
	COMMIT
	UNLOCK(A)
$B := \frac{B}{A}$	
$\Downarrow$	
ABORT	

Példa piszkos adatból eredő hibára zárolásos ütemezés esetén (persze időbélyegekkel is van ilyen):

Ha az osztáskor A értéke éppen 0, akkor T_1 ABORT-ál, és emiatt sok baj lesz:

- B-n zár marad, ezt fel kell oldani
- T_1 félig futott csak le, amit eddig számolt, azt vissza kell csinálni

T_1	T_2
LOCK(A)	
READ(A)	
A:=A+100	
WRITE(A)	
LOCK(B)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	$A := A \cdot 25$
READ(B)	
	WRITE(A)
	COMMIT
	UNLOCK(A)
$B := \frac{B}{A}$	
$\Downarrow$	
ABORT	

Példa piszkos adatból eredő hibára zárolásos ütemezés esetén (persze időbélyegekkel is van ilyen):

Ha az osztáskor A értéke éppen 0, akkor T_1 ABORT-ál, és emiatt sok baj lesz:

- B-n zár marad, ezt fel kell oldani
- T_1 félig futott csak le, amit eddig számolt, azt vissza kell csinálni
- T_2 rossz adatot olvasott (mert a T_1 által A-ba beleírt értéket visszavontuk), így T_2-t is vissza kell csinálni

T_1	T_2
LOCK(A)	
READ(A)	
A:=A+100	
WRITE(A)	
LOCK(B)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	$A := A \cdot 25$
READ(B)	
	WRITE(A)
	COMMIT
	UNLOCK(A)
$B := \frac{B}{A}$	
$\Downarrow$	
ABORT	

Példa piszkos adatból eredő hibára zárolásos ütemezés esetén (persze időbélyegekkel is van ilyen):

Ha az osztáskor A értéke éppen 0, akkor T_1 ABORT-ál, és emiatt sok baj lesz:

- B-n zár marad, ezt fel kell oldani
- T_1 félig futott csak le, amit eddig számolt, azt vissza kell csinálni
- T_2 rossz adatot olvasott (mert a T_1 által A-ba beleírt értéket visszavontuk), így T_2-t is vissza kell csinálni

Összességében ebben az esetben T_1 és T_2 minden hatását ki kell irtani a DB-ből.

T_1	T_2
LOCK(A)	
READ(A)	
A:=A+100	
WRITE(A)	
LOCK(B)	
UNLOCK(A)	
	LOCK(A)
	READ(A)
	$A := A \cdot 25$
READ(B)	
	WRITE(A)
	COMMIT
	UNLOCK(A)
$B := \frac{B}{A}$	
$\Downarrow$	
ABORT	

Példa piszkos adatból eredő hibára zárolásos ütemezés esetén (persze időbélyegekkel is van ilyen):
Ha az osztáskor A értéke éppen 0, akkor T_1 ABORT-ál, és emiatt sok baj lesz:

- B-n zár marad, ezt fel kell oldani
- T_1 félig futott csak le, amit eddig számolt, azt vissza kell csinálni
- T_2 rossz adatot olvasott (mert a T_1 által A-ba beleírt értéket visszavontuk), így T_2-t is vissza kell csinálni

Összességében ebben az esetben T_1 és T_2 minden hatását ki kell irtani a DB-ből.
Ha esetleg közben még mások is olvasták a T_1 vagy a T_2 által írt értékeket, akkor **lavina**: egymás után kell ABORT-okat elrendelni a tranzakcióknál piszkos adatból eredő hiba miatt.

Megoldások piszkos adat és lavina ellen

Különböző megoldások a tranzakcióhibákból (programhiba vagy rendszer általi ABORT) származó problémákra:

- Olyan tranzakciótól, ami nem COMMIT-ált, nem olvasunk. (Nem olvasunk olyan értéket, amit olyan tranzakció írt, aminek még nem volt COMMIT).

Megoldások piszkos adat és lavina ellen

Különböző megoldások a tranzakcióhibákból (programhiba vagy rendszer általi ABORT) származó problémákra:

- Olyan tranzakciótól, ami nem COMMIT-ált, nem olvasunk. (Nem olvasunk olyan értéket, amit olyan tranzakció írt, aminek még nem volt COMMIT).
- Hagyjuk, hogy minden tranzakció azt csináljon, amit akar, ha lavina lesz, akkor majd megoldjuk (UNDO protokoll, nem lesz részletesen, de létezik)

Megoldások piszkos adat és lavina ellen

Különböző megoldások a tranzakcióhibákból (programhiba vagy rendszer általi ABORT) származó problémákra:

- Olyan tranzakciótól, ami nem COMMIT-ált, nem olvasunk. (Nem olvasunk olyan értéket, amit olyan tranzakció írt, aminek még nem volt COMMIT).
- Hagyjuk, hogy minden tranzakció azt csináljon, amit akar, ha lavina lesz, akkor majd megoldjuk (UNDO protokoll, nem lesz részletesen, de létezik)
- Zárolási protokollt kényszerítünk a tranzakciókra, ami biztosítja, hogy nem lesz piszkos adatból probléma, lavina:

  szigorú 2PL:
  - 2PL

Megoldások piszkos adat és lavina ellen

Különböző megoldások a tranzakcióhibákból (programhiba vagy rendszer általi ABORT) származó problémákra:

- Olyan tranzakciótól, ami nem COMMIT-ált, nem olvasunk. (Nem olvasunk olyan értéket, amit olyan tranzakció írt, aminek még nem volt COMMIT).
- Hagyjuk, hogy minden tranzakció azt csináljon, amit akar, ha lavina lesz, akkor majd megoldjuk (UNDO protokoll, nem lesz részletesen, de létezik)
- Zárolási protokollt kényszerítünk a tranzakciókra, ami biztosítja, hogy nem lesz piszkos adatból probléma, lavina:

  szigorú 2PL:
  - 2PL
  - DB-be írás csak COMMIT után

Megoldások piszkos adat és lavina ellen

Különböző megoldások a tranzakcióhibákból (programhiba vagy rendszer általi ABORT) származó problémákra:

- Olyan tranzakciótól, ami nem COMMIT-ált, nem olvasunk. (Nem olvasunk olyan értéket, amit olyan tranzakció írt, aminek még nem volt COMMIT).
- Hagyjuk, hogy minden tranzakció azt csináljon, amit akar, ha lavina lesz, akkor majd megoldjuk (UNDO protokoll, nem lesz részletesen, de létezik)
- Zárolási protokollt kényszerítünk a tranzakciókra, ami biztosítja, hogy nem lesz piszkos adatból probléma, lavina:

  szigorú 2PL:
  - 2PL
  - DB-be írás csak COMMIT után
  - zárak elengedése csak írás után

Szigorú 2PL protokoll

Tétel

Ha mindegyik tranzakció a szigorú 2PL protokollt követi, akkor az ütemezés sorosítható lesz és lavinamentes.

Bizonyítás.

Mivel a tranzakciók követik a 2PL protokollt, ezért az ütemezés sorosítható lesz.

Szigorú 2PL protokoll

Tétel

Ha mindegyik tranzakció a szigorú 2PL protokollt követi, akkor az ütemezés sorosítható lesz és lavinamentes.

Bizonyítás.

Mivel a tranzakciók követik a 2PL protokollt, ezért az ütemezés sorosítható lesz. Azért lesz lavinamentes is, mert egy T_i tranzakció csak akkor olvashatja egy másik T_j tranzakció írását, ha T_j már elengedte a zárakat, de az meg csak COMMIT után lehet, amikor T_j már biztos nem száll el.

Szigorú 2PL protokoll

Tétel

Ha mindegyik tranzakció a szigorú 2PL protokollt követi, akkor az ütemezés sorosítható lesz és lavinamentes.

Bizonyítás.

Mivel a tranzakciók követik a 2PL protokollt, ezért az ütemezés sorosítható lesz. Azért lesz lavinamentes is, mert egy T_i tranzakció csak akkor olvashatja egy másik T_j tranzakció írását, ha T_j már elengedte a zárakat, de az meg csak COMMIT után lehet, amikor T_j már biztos nem száll el. □

Megjegyzések:

1. Elég az írások, a COMMIT és a zárkérések sorrendjét figyelni, ahhoz hogy jó ütemezés legyen és ráadásul ezt minden tranzakció meg tudja maga tenni, nem kell a többire figyelnie.

Szigorú 2PL protokoll

Tétel

Ha mindegyik tranzakció a szigorú 2PL protokollt követi, akkor az ütemezés sorosítható lesz és lavinamentes.

Bizonyítás.

Mivel a tranzakciók követik a 2PL protokollt, ezért az ütemezés sorosítható lesz. Azért lesz lavinamentes is, mert egy T_i tranzakció csak akkor olvashatja egy másik T_j tranzakció írását, ha T_j már elengedte a zárakat, de az meg csak COMMIT után lehet, amikor T_j már biztos nem száll el. □

Megjegyzések:

1. Elég az írások, a COMMIT és a zárkérések sorrendjét figyelni, ahhoz hogy jó ütemezés legyen és ráadásul ezt minden tranzakció meg tudja maga tenni, nem kell a többire figyelnie.

2. Mivel írás csak COMMIT után van, nem kell azzal sem bajlódni, hogy visszagörgessük az elszállt tranzakciókat, mert ezeknek még úgysem látszik semmi hatásuk.

Adatbázisok elmélete

Rendszerhibák utáni helyreállítás, UNDO protokoll

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

25. előadás

Rendszerhibák utáni helyreállítás

Az eddigiekben azzal foglalkoztunk, hogy a tranzakciók hibái esetén mit lehet tenni. Erre a szigorú 2PL jó megoldást nyújt, de mi van a komolyabb hibák, a rendszerhibák esetén?

Rendszerhibák utáni helyreállítás

Az eddigiekben azzal foglalkoztunk, hogy a tranzakciók hibái esetén mit lehet tenni.
Erre a szigorú 2PL jó megoldást nyújt, de mi van a komolyabb hibák, a rendszerhibák esetén?
Azt mindig feltesszük, hogy a háttértár nem sérül, csak a belső memória, a puffer egy része száll el.

Rendszerhibák utáni helyreállítás

Az eddigiekben azzal foglalkoztunk, hogy a tranzakciók hibái esetén mit lehet tenni.
Erre a szigorú 2PL jó megoldást nyújt, de mi van a komolyabb hibák, a rendszerhibák esetén?
Azt mindig feltesszük, hogy a háttértár nem sérül, csak a belső memória, a puffer egy része száll el.
A belső tár sérülése elleni védekezés két részből áll:

1. Felkészülés a hibára: **naplózás**

Rendszerhibák utáni helyreállítás

Az eddigiekben azzal foglalkoztunk, hogy a tranzakciók hibái esetén mit lehet tenni.
Erre a szigorú 2PL jó megoldást nyújt, de mi van a komolyabb hibák, a rendszerhibák esetén?
Azt mindig feltesszük, hogy a háttértár nem sérül, csak a belső memória, a puffer egy része száll el.
A belső tár sérülése elleni védekezés két részből áll:

1. Felkészülés a hibára: **naplózás**
2. Hiba után helyreállítás: **a napló segítségével egy konzisztens állapot helyreállítása**

Rendszerhibák utáni helyreállítás

Az eddigiekben azzal foglalkoztunk, hogy a tranzakciók hibái esetén mit lehet tenni. Erre a szigorú 2PL jó megoldást nyújt, de mi van a komolyabb hibák, a rendszerhibák esetén?
Azt mindig feltesszük, hogy a háttértár nem sérül, csak a belső memória, a puffer egy része száll el.
A belső tár sérülése elleni védekezés két részből áll:

1. Felkészülés a hibára: **naplózás**
2. Hiba után helyreállítás: **a napló segítségével egy konzisztens állapot helyreállítása**

Természetesen a naplózás és a hiba utáni helyreállítás összhangban vannak, de van több különböző naplózási protokoll (és ennek megfelelő helyreállítás).

Napló

A tranzakciók legfontosabb történéseit írjuk ide, például:

- T_i kezd: (T_i, BEGIN)

Napló

A tranzakciók legfontosabb történéseit írjuk ide, például:

- T_i kezd: (T_i, BEGIN)
- T_i írja A-t: (T_i, A, régi érték, új érték)
  (néha elég csak a régi vagy csak az új érték, a naplózási protokolltól függően)

Napló

A tranzakciók legfontosabb történéseit írjuk ide, például:

- T_i kezd: (T_i, BEGIN)
- T_i írja A-t: (T_i, A, régi érték, új érték)
  (néha elég csak a régi vagy csak az új érték, a naplózási protokolltól függően)
- T_i COMMIT-ál: (T_i, COMMIT)

Napló

A tranzakciók legfontosabb történéseit írjuk ide, például:

- T_i kezd: (T_i, BEGIN)
- T_i írja A-t: (T_i, A, régi érték, új érték)
  (néha elég csak a régi vagy csak az új érték, a naplózási protokolltól függően)
- T_i COMMIT-ál: (T_i, COMMIT)
- T_i ABORT-ál: (T_i, ABORT)

Napló

A tranzakciók legfontosabb történéseit írjuk ide, például:

- T_i kezd: (T_i, BEGIN)
- T_i írja A-t: (T_i, A, régi érték, új érték)
  (néha elég csak a régi vagy csak az új érték, a naplózási protokolltól függően)
- T_i COMMIT-ál: (T_i, COMMIT)
- T_i ABORT-ál: (T_i, ABORT)

A napló időrendben tartalmazza a történéseket és tipikusan a háttértáron tartjuk, amiről feltesszük, hogy nem sérül.

Napló

A tranzakciók legfontosabb történéseit írjuk ide, például:

- T_i kezd: (T_i, BEGIN)
- T_i írja A-t: (T_i, A, régi érték, új érték)
  (néha elég csak a régi vagy csak az új érték, a naplózási protokolltól függően)
- T_i COMMIT-ál: (T_i, COMMIT)
- T_i ABORT-ál: (T_i, ABORT)

A napló időrendben tartalmazza a történéseket és tipikusan a háttértáron tartjuk, amiről feltesszük, hogy nem sérül.
Fontos, hogy a naplóbejegyzéseket mikor írjuk át a pufferből a lemezre.

UNDO protokoll-naplózás

Fő szabályok:

- Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.

UNDO protokoll-naplózás

Fő szabályok:

- Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
- Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

UNDO protokoll-naplózás

Fő szabályok:

- Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
- Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

UNDO protokoll

A (nem teljesen szigorú) 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: (T_i, BEGIN), (T_i, A régi érték) vagy (T_i, ABORT)

UNDO protokoll-naplózás

Fő szabályok:

- Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
- Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

UNDO protokoll

A (nem teljesen szigorú) 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: (T_i, BEGIN), (T_i, A régi érték) vagy (T_i, ABORT)
2. Tényleges írás az adatbázisba a háttértáron, nem a pufferben: OUTPUT(A)

UNDO protokoll-naplózás

Fő szabályok:

- Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
- Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

UNDO protokoll

A (nem teljesen szigorú) 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: (T_i, BEGIN), (T_i, A régi érték) vagy (T_i, ABORT)
2. Tényleges írás az adatbázisba a háttértáron, nem a pufferben: OUTPUT(A)
3. COMMIT után a napló háttértárra írása.

UNDO protokoll-naplózás

Fő szabályok:

- Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
- Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

UNDO protokoll

A (nem teljesen szigorú) 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: (T_i, BEGIN), (T_i, A régi érték) vagy (T_i, ABORT)
2. Tényleges írás az adatbázisba a háttértáron, nem a pufferben: OUTPUT(A)
3. COMMIT után a napló háttértárra írása.
4. Zárak elengedése

Megjegyzések

- nincs lavina, mert zárelengedés csak COMMIT után (azaz piszkos adatot nem olvashatunk)

Megjegyzések

- nincs lavina, mert zárelengedés csak COMMIT után (azaz piszkos adatot nem olvashatunk)
- sorosítható, mert 2PL

Megjegyzések

- nincs lavina, mert zárelengedés csak COMMIT után (azaz piszkos adatot nem olvashatunk)
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt az UNDO helyreállítás

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T, BEGIN)

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)
READ(B, t)	8	16	8	8	8	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 8)

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 8)
FLUSH LOG						

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 8)$
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 8)$
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 8)
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	$(T$, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 8)$
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 8)$
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						$(T$, COMMIT)

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	$(T$, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 8)$
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 8)$
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						$(T$, COMMIT)
FLUSH LOG						

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 8)
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						(T, COMMIT)
FLUSH LOG						
UNLOCK(A)						

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 8)
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						(T, COMMIT)
FLUSH LOG						
UNLOCK(A)						
UNLOCK(B)						

FLUSH LOG: napló kiírása a háttértárra t: lokális változó

A_M, B_M, A_D, B_D: Az A és B cellák tartalma memóriában illetve a lemezen.

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT)

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT)

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani
- Második feladat: visszaállítás
  A napló végétől visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely T_i-re találtunk (T_i, COMMIT) és (T_i, ABORT) bejegyzéseket.

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani
- Második feladat: visszaállítás
  A napló végétől visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely T_i-re találtunk (T_i, COMMIT) és (T_i, ABORT) bejegyzéseket.
  Ha van egy (T_i, X, v) bejegyzés:
  - Ha láttunk már (T_i, COMMIT) bejegyzést (visszafelé haladva), akkor T_i már befejeződött, értékét kiírtuk a tárra

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani
- Második feladat: visszaállítás
  A napló végétől visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely T_i-re találtunk (T_i, COMMIT) és (T_i, ABORT) bejegyzéseket.
  Ha van egy (T_i, X, v) bejegyzés:
  - Ha láttunk már (T_i, COMMIT) bejegyzést (visszafelé haladva), akkor T_i már befejeződött, értékét kiírtuk a tárra $\Longrightarrow$ nem csinálunk semmit

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani
- Második feladat: visszaállítás
  A napló végétől visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely T_i-re találtunk (T_i, COMMIT) és (T_i, ABORT) bejegyzéseket.
  Ha van egy (T_i, X, v) bejegyzés:
  - Ha láttunk már (T_i, COMMIT) bejegyzést (visszafelé haladva), akkor T_i már befejeződött, értékét kiírtuk a tárra $\Longrightarrow$ nem csinálunk semmit
  - Minden más esetben (vagy volt (T_i, ABORT) vagy semmi)

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani
- Második feladat: visszaállítás
  A napló végétől visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely T_i-re találtunk (T_i, COMMIT) és (T_i, ABORT) bejegyzéseket.
  Ha van egy (T_i, X, v) bejegyzés:
  - Ha láttunk már (T_i, COMMIT) bejegyzést (visszafelé haladva), akkor T_i már befejeződött, értékét kiírtuk a tárra $\Longrightarrow$ nem csinálunk semmit
  - Minden más esetben (vagy volt (T_i, ABORT) vagy semmi) $\Longrightarrow$ X-be visszaírjuk v-t

UNDO helyreállítás

Ha hiba történt $\Longrightarrow$ konzisztens állapot visszaállítása
$\Longrightarrow$ nem befejezett tranzakciók hatásának törlése

- Első feladat: Eldönteni melyek a sikeresen befejezett és melyek nem befejezett tranzakciók.
  - Ha van (T, BEGIN) és van (T, COMMIT) $\Longrightarrow$ minden változás a lemezen van $\surd$
  - Ha van (T, BEGIN), de nincs (T, COMMIT) $\Longrightarrow$ lehet olyan változás, ami nem került még a lemezre, de lehet olyan is ami kikerült $\Longrightarrow$ ezeket vissza kell állítani
- Második feladat: visszaállítás
  A napló végétől visszafelé (pontosabban a hibától) haladva: megjegyezzük, hogy mely T_i-re találtunk (T_i, COMMIT) és (T_i, ABORT) bejegyzéseket.
  Ha van egy (T_i, X, v) bejegyzés:
  - Ha láttunk már (T_i, COMMIT) bejegyzést (visszafelé haladva), akkor T_i már befejeződött, értékét kiírtuk a tárra $\Longrightarrow$ nem csinálunk semmit
  - Minden más esetben (vagy volt (T_i, ABORT) vagy semmi) $\Longrightarrow$ X-be visszaírjuk v-t
- Harmadik feladat: Ha végeztünk, minden nem teljes T_i-re írjunk (T_i, ABORT) a napló végére.

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 8)$
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 8)$
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						(T, COMMIT)
(FLUSH LOG)						
UNLOCK(A)						
UNLOCK(B)						

Megjegyzések

- Mi van ha a helyreállítás közben hiba történik? Már bizonyos értékeket visszaállítottunk, de utána elakadunk.

Megjegyzések

- Mi van ha a helyreállítás közben hiba történik? Már bizonyos értékeket visszaállítottunk, de utána elakadunk.
  $\Longrightarrow$ Kezdjük előről a visszaállítást! Ha már valami vissza volt állítva, legfeljebb még egyszer „visszaállítjuk”

Megjegyzések

- Mi van ha a helyreállítás közben hiba történik? Már bizonyos értékeket visszaállítottunk, de utána elakadunk.
  $\Longrightarrow$ Kezdjük előről a visszaállítást! Ha már valami vissza volt állítva, legfeljebb még egyszer „visszaállítjuk” $\Longrightarrow$ nem történik semmi.

Megjegyzések

- Mi van ha a helyreállítás közben hiba történik? Már bizonyos értékeket visszaállítottunk, de utána elakadunk.
  $\Longrightarrow$ Kezdjük előről a visszaállítást! Ha már valami vissza volt állítva, legfeljebb még egyszer „visszaállítjuk” $\Longrightarrow$ nem történik semmi.

- **Ez így nagyon sokáig tarthat, mert el kell mennünk a napló elejéig.**

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**.

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy (CHECKPOINT)

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy (CHECKPOINT)
5. A naplót is háttértárra írjuk: (FLUSH LOG)

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy (CHECKPOINT)
5. A naplót is háttértárra írjuk: (FLUSH LOG)

Ezután nyilván elég az első CHECKPOINT-ig visszamenni, hiszen előtte minden T_i már valahogy befejeződött.

CHECKPOINT képzése

Meddig menjünk vissza a naplóban? Honnan tudjuk, hogy mikor vagyunk egy biztosan konzisztens állapotnál?
Erre való a **CHECKPOINT**. Ennek képzése:

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy (CHECKPOINT)
5. A naplót is háttértárra írjuk: (FLUSH LOG)

Ezután nyilván elég az első CHECKPOINT-ig visszamenni, hiszen előtte minden T_i már valahogy befejeződött.
$\Longrightarrow$ Teljesen le kell állítani a rendszert.

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Megvárjuk, hogy minden fenti T_i végetérjen. (Közben más tranzakciók elindulhatnak.)

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Megvárjuk, hogy minden fenti T_i végetérjen. (Közben más tranzakciók elindulhatnak.)
4. Ha mind befejeződött: (END CHECKPOINT) és (FLUSH LOG)

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Megvárjuk, hogy minden fenti T_i végetérjen. (Közben más tranzakciók elindulhatnak.)
4. Ha mind befejeződött: (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Megvárjuk, hogy minden fenti T_i végetérjen. (Közben más tranzakciók elindulhatnak.)
4. Ha mind befejeződött: (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig.

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Megvárjuk, hogy minden fenti T_i végetérjen. (Közben más tranzakciók elindulhatnak.)
4. Ha mind befejeződött: (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig.
- Ha előbb (START CHECKPOINT ($T_1, \ldots, T_k$))-ot találunk $\Longrightarrow$ ezek nem mindegyike fejeződött be (meg esetleg mások sem, amik még később kezdődtek)

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Megvárjuk, hogy minden fenti T_i végetérjen. (Közben más tranzakciók elindulhatnak.)
4. Ha mind befejeződött: (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig.
- Ha előbb (START CHECKPOINT ($T_1, \ldots, T_k$))-ot találunk $\Longrightarrow$ ezek nem mindegyike fejeződött be (meg esetleg mások sem, amik még később kezdődtek) $\Longrightarrow$ elég visszamenni a legkorábban kezdődött T_i elejére

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$
(T_2, COMMIT)
(END CHECKPOINT)

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$
(T_2, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$
(T_2, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$ $\leftarrow$

- T_3 nem fejeződött be

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$
(T_2, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$ $\leftarrow$

- T_3 nem fejeződött be
  $\Longrightarrow F \rightarrow 30$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$ $\leftarrow$
(T_2, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$ $\leftarrow$

- T_3 nem fejeződött be
  $\Longrightarrow F \rightarrow 30$
- T_3 nem fejeződött be

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$ $\leftarrow$
(T_2, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow F \rightarrow 30$
- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
$(\text{START CHECKPOINT } (T_1, T_2))$ $\leftarrow$
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$ $\leftarrow$
(T_2, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$ $\leftarrow$

- T_3 nem fejeződött be
  $\Longrightarrow F \rightarrow 30$
- T_3 nem fejeződött be
  $\Longrightarrow E \rightarrow 25$
- (START CHECKPOINT)

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
$(\text{START CHECKPOINT } (T_1, T_2))$ $\leftarrow$
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$ $\leftarrow$
(T_2, COMMIT)
(END CHECKPOINT)
$(T_3, F, 30)$ $\leftarrow$

- T_3 nem fejeződött be
  $\Longrightarrow F \rightarrow 30$
- T_3 nem fejeződött be
  $\Longrightarrow E \rightarrow 25$
- (START CHECKPOINT)
  $\Longrightarrow \surd$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)

Példa

(T_1, BEGIN)
(T_1, A, 5)
(T_2, BEGIN)
(T_2, B, 10)
(START CHECKPOINT (T_1, T_2))
(T_2, C, 15)
(T_3, BEGIN)
(T_1, D, 20)
(T_1, COMMIT)

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
$(\text{START CHECKPOINT } (T_1, T_2))$
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT)
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be
  $\Longrightarrow E \rightarrow 25$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
$(\text{START CHECKPOINT } (T_1, T_2))$
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
$(T_1, \text{COMMIT}) \leftarrow$
$(T_3, E, 25) \leftarrow$

- T_3 nem fejeződött be
  $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$
(T_3, BEGIN)
$(T_1, D, 20)$
$(T_1, \text{COMMIT}) \leftarrow$
$(T_3, E, 25) \leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
$(\text{START CHECKPOINT } (T_1, T_2))$
$(T_2, C, 15)$ $\leftarrow$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT) $\leftarrow$
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk
- T_2 nem fejeződött be

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2))
$(T_2, C, 15)$ $\leftarrow$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT) $\leftarrow$
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk
- T_2 nem fejeződött be $\Longrightarrow C \rightarrow 15$

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
(START CHECKPOINT (T_1, T_2)) $\leftarrow$
$(T_2, C, 15)$ $\leftarrow$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT) $\leftarrow$
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk
- T_2 nem fejeződött be $\Longrightarrow C \rightarrow 15$
- (START CHECKPOINT)

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$
$(\text{START CHECKPOINT } (T_1, T_2)) \leftarrow$
$(T_2, C, 15) \leftarrow$
(T_3, BEGIN)
$(T_1, D, 20)$
$(T_1, \text{COMMIT}) \leftarrow$
$(T_3, E, 25) \leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk
- T_2 nem fejeződött be $\Longrightarrow C \rightarrow 15$
- (START CHECKPOINT) $\Longrightarrow$ elég visszamenni T_2 elejéig

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$ $\leftarrow$
(START CHECKPOINT (T_1, T_2)) $\leftarrow$
$(T_2, C, 15)$ $\leftarrow$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT) $\leftarrow$
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk
- T_2 nem fejeződött be $\Longrightarrow C \rightarrow 15$
- (START CHECKPOINT) $\Longrightarrow$ elég visszamenni T_2 elejéig
- T_2 nem fejeződött be

Példa

(T_1, BEGIN)
$(T_1, A, 5)$
(T_2, BEGIN)
$(T_2, B, 10)$ $\leftarrow$
(START CHECKPOINT (T_1, T_2)) $\leftarrow$
$(T_2, C, 15)$ $\leftarrow$
(T_3, BEGIN)
$(T_1, D, 20)$
(T_1, COMMIT) $\leftarrow$
$(T_3, E, 25)$ $\leftarrow$

- T_3 nem fejeződött be $\Longrightarrow E \rightarrow 25$
- T_1 befejeződött $\Longrightarrow$ T_1-et nem bántjuk
- T_2 nem fejeződött be $\Longrightarrow C \rightarrow 15$
- (START CHECKPOINT) $\Longrightarrow$ elég visszamenni T_2 elejéig
- T_2 nem fejeződött be $\Longrightarrow B \rightarrow 10$

Adatbázisok elmélete

REDO protokoll

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

26. előadás

REDO protokoll-naplózás

Fő szabály:

- Mielőtt a lemezen módosítunk egy X adatelemet, a (T, X, v) és a (T, COMMIT) bejegyzést is ki kell írnunk a naplóba.

REDO protokoll-naplózás

Fő szabály:

- Mielőtt a lemezen módosítunk egy X adatelemet, a (T, X, v) és a $(T$, COMMIT) bejegyzést is ki kell írnunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: $(T_i$, BEGIN), $(T_i, A$, új érték), $(T_i$, ABORT)

REDO protokoll-naplózás

Fő szabály:

- Mielőtt a lemezen módosítunk egy X adatelemet, a (T, X, v) és a $(T$, COMMIT) bejegyzést is ki kell írnunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: $(T_i$, BEGIN), $(T_i, A$, új érték), $(T_i$, ABORT)
2. COMMIT után a napló háttértárra írása

REDO protokoll-naplózás

Fő szabály:

- Mielőtt a lemezen módosítunk egy X adatelemet, a (T, X, v) és a $(T$, COMMIT) bejegyzést is ki kell írnunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: $(T_i$, BEGIN), $(T_i, A$, új érték), $(T_i$, ABORT)
2. COMMIT után a napló háttértárra írása
3. Tényleges írás az adatbázisba a háttértáron, nem a pufferben

REDO protokoll-naplózás

Fő szabály:

- Mielőtt a lemezen módosítunk egy X adatelemet, a (T, X, v) és a $(T$, COMMIT) bejegyzést is ki kell írnunk a naplóba.

REDO protokoll

Ez a szigorú 2PL kiegészítése, vagyis a zárkérések 2PL szerint történnek, ezen felül pedig a műveletek és ezek naplózása az alábbi sorrendben történik:

1. A tranzakciók történéseinek feljegyzése a naplóba, a belső táron: $(T_i$, BEGIN), $(T_i, A$, új érték), $(T_i$, ABORT)
2. COMMIT után a napló háttértárra írása
3. Tényleges írás az adatbázisba a háttértáron, nem a pufferben
4. Zárak elengedése

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- Különbség a az UNDO protokollhoz képest:
  - Az adat változás utáni értékét jegyezzük fel a naplóba

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- Különbség a az UNDO protokollhoz képest:
  - Az adat változás utáni értékét jegyezzük fel a naplóba
  - Máshová rakjuk a COMMIT-ot, a kiírás elé

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- Különbség a az UNDO protokollhoz képest:
  - Az adat változás utáni értékét jegyezzük fel a naplóba
  - Máshová rakjuk a COMMIT-ot, a kiírás elé $\Longrightarrow$ megtelhet a puffer

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- Különbség a az UNDO protokollhoz képest:
  - Az adat változás utáni értékét jegyezzük fel a naplóba
  - Máshová rakjuk a COMMIT-ot, a kiírás elé $\Longrightarrow$ megtelhet a puffer
  - Az UNDO protokoll esetleg túl gyakran akar írni

Megjegyzések:

- nincs lavina, mert zárelengedés csak COMMIT után
- sorosítható, mert 2PL
- vissza lehet hozni konzisztens állapotba a DB-t, akkor is, ha a belső tár sérül, erre lesz majd mindjárt a REDO helyreállítás
- Különbség a az UNDO protokollhoz képest:
  - Az adat változás utáni értékét jegyezzük fel a naplóba
  - Máshová rakjuk a COMMIT-ot, a kiírás elé $\Longrightarrow$ megtelhet a puffer
  - Az UNDO protokoll esetleg túl gyakran akar írni $\Longrightarrow$ itt el lehet halasztani az írást

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	$(T$, BEGIN$)$
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	$(T, A, 16)$
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	$(T, B, 16)$
						$(T$, COMMIT$)$
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)
3. Addig megyünk vissza a naplóban, amíg biztosan konzisztens állapotot nem találunk (eleje vagy CHECKPOINT)

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)
3. Addig megyünk vissza a naplóban, amíg biztosan konzisztens állapotot nem találunk (eleje vagy CHECKPOINT)
4. A COMMIT-tált tranzakciók írásait előlről kezdve (a legelső COMMIT-ált elejétől) megismételjük (ha már egyszer be volt írva, az se baj, akkor csak felülírjuk ugyanazzal). Ezt meg tudjuk tenni, mert ismerjük az új értékeket.

REDO helyreállítás

Ha rendszerhiba történt és megsérült a belső tár, akkor az alábbiakat tesszük:

1. Minden zárat feloldunk
2. A napló mentett részét nézzük visszafele, megkeressük azokat a tranzakciókat, amikre volt már COMMIT (a többi nem érdekes, mert ha még nem volt a COMMIT-juk kimentve, akkor nem is írtak a DB-be)
3. Addig megyünk vissza a naplóban, amíg biztosan konzisztens állapotot nem találunk (eleje vagy CHECKPOINT)
4. A COMMIT-tált tranzakciók írásait előlről kezdve (a legelső COMMIT-ált elejétől) megismételjük (ha már egyszer be volt írva, az se baj, akkor csak felülírjuk ugyanazzal). Ezt meg tudjuk tenni, mert ismerjük az új értékeket.
5. Minden nem befejezett T_i tranzakcióra (T_i,ABORT)-ot írunk a napló végére, (FLUSH LOG)

Megjegyzések a REDO helyreállításhoz

- Ha a napló háttértáron van, akkor mindent újra tudunk csinálni, ami meg még nem került ki, azzal kapcsolatban változtatás se történt, nem kell visszacsinálni semmit.

Megjegyzések a REDO helyreállításhoz

- Ha a napló háttértáron van, akkor mindent újra tudunk csinálni, ami meg még nem került ki, azzal kapcsolatban változtatás se történt, nem kell visszacsinálni semmit.
- Ha a helyreállítás során lenne újra hiba, akkor a napló marad, mert az már kint van, ez alapján újra kezdhetjük a helyreállítást.

Megjegyzések a REDO helyreállításhoz

- Ha a napló háttértáron van, akkor mindent újra tudunk csinálni, ami meg még nem került ki, azzal kapcsolatban változtatás se történt, nem kell visszacsinálni semmit.
- Ha a helyreállítás során lenne újra hiba, akkor a napló marad, mert az már kint van, ez alapján újra kezdhetjük a helyreállítást.
- Eredmény: a háttértárra kikerült COMMIT-oknak megfelelő tranazakciók eredménye látszik, a többiekéből pedig semmi.

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 16)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 16)
						(T, COMMIT)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

Példa

T_1	napló
	(T_1, BEGIN)
LOCK(A)	
LOCK(B)	
	(T_1, A, x)
	(T_1, B, y)
	(T_1, COMMIT)
WRITE(A)	
WRITE(B)	
UNLOCK(A)	
UNLOCK(B)	

(T_1, A, x) jelentése: T_1 A-ba x-et írja
Ekkor a tényleges írás nem történik meg, csak a naplóba kerül ez bele, a tényleges írás csak a COMMIT után jön.

Példa

T_1	napló
	(T_1, BEGIN)
LOCK(A)	
LOCK(B)	
	(T_1, A, x)
	(T_1, B, y)
	(T_1, COMMIT)
WRITE(A)	
WRITE(B)	
UNLOCK(A)	
UNLOCK(B)	

(T_1, A, x) jelentése: T_1 A-ba x-et írja
Ekkor a tényleges írás nem történik meg, csak a naplóba kerül ez bele, a tényleges írás csak a COMMIT után jön.

Ha a belső tár hibája a COMMIT háttértárra írása előtt történik, akkor még semmi valódi írás nem volt, azaz semmit se kell csinálni. Ha azonban a COMMIT után van, akkor a naplóban megvan minden utasítás, újra meg lehet csinálni T_1-et.

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy CHECKPOINT

CHECKPOINT képzése

1. Megtiltjuk új tranzakció indítását
2. Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér
3. Minden puffert a háttértárra írunk, ekkor az adatbázis állapota biztosan konzisztens lesz
4. A naplóba beírjuk, hogy CHECKPOINT
5. A naplót is háttértárra írjuk

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig.

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig. $\Longrightarrow$ innen előre minden itt szereplő T_i-re és minden később kezdődő más tranzakcióra REDO

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig. $\Longrightarrow$ innen előre minden itt szereplő T_i-re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT ($T_1, \ldots, T_k$))-ot találunk

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig. $\Longrightarrow$ innen előre minden itt szereplő T_i-re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT ($T_1, \ldots, T_k$))-ot találunk $\Longrightarrow$ ezek nem mindegyike írta ki adatai (meg esetleg mások sem, amik még később kezdődtek)

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig. $\Longrightarrow$ innen előre minden itt szereplő T_i-re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT ($T_1, \ldots, T_k$))-ot találunk $\Longrightarrow$ ezek nem mindegyike írta ki adatai (meg esetleg mások sem, amik még később kezdődtek) $\Longrightarrow$ elég visszamenni az előző (START CHECKPOINT)-hoz

CHECKPOINT képzése működés közben

1. A naplóba beírjuk, hogy (START CHECKPOINT ($T_1, \ldots, T_k$)), ahol T_i az összes éppen aktív tranzakció
2. A naplót háttértárra írjuk: FLUSH LOG
3. Az összes olyan adatelemet kiírjuk a lemezre, amit olyan tranzakciók indítottak, amik még a CHECKPOINT előtt befejeződtek, de még nem írtak ki mindent a lemezre.
4. (END CHECKPOINT) és (FLUSH LOG)

Visszaállítás

- Visszafelé olvasva, ha előbb (END CHECKPOINT) van $\Longrightarrow$ elég visszamenni a következő START CHECKPOINT-ig. $\Longrightarrow$ innen előre minden itt szereplő T_i-re és minden később kezdődő más tranzakcióra REDO
- Ha előbb (START CHECKPOINT ($T_1, \ldots, T_k$))-ot találunk $\Longrightarrow$ ezek nem mindegyike írta ki adatai (meg esetleg mások sem, amik még később kezdődtek) $\Longrightarrow$ elég visszamenni az előző (START CHECKPOINT)-hoz $\Longrightarrow$ onnan előre REDO

Előnyök, hátrányok

A CHECKPOINT ütemezése:

- adott idő letelte után

Előnyök, hátrányok

A CHECKPOINT ütemezése:

- adott idő letelte után
- adott lefutott tranzakció után

Előnyök, hátrányok

A CHECKPOINT ütemezése:

- adott idő letelte után
- adott lefutott tranzakció után

Ha ritkák a rendszerhibák, elég ritka CHECKPOINT.

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után azonnal kiírjuk a lemezre az értékeket

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után azonnal kiírjuk a lemezre az értékeket $\Longrightarrow$ sok írás

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után azonnal kiírjuk a lemezre az értékeket $\Longrightarrow$ sok írás
- REDO hátránya: Nem írunk, amíg nincs COMMIT

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után azonnal kiírjuk a lemezre az értékeket $\Longrightarrow$ sok írás
- REDO hátránya: Nem írunk, amíg nincs COMMIT $\Longrightarrow$ nagy memóriaigény

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után azonnal kiírjuk a lemezre az értékeket $\Longrightarrow$ sok írás
- REDO hátránya: Nem írunk, amíg nincs COMMIT $\Longrightarrow$ nagy memóriaigény

UNDO/REDO

Fő elv:

- Mielőtt az adatbázis bármely X elemének értékét a lemezen módosítanánk, a (T, X, v, w) naplóbejegyzésnek a lemezre kell kerülnie.

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után azonnal kiírjuk a lemezre az értékeket $\Longrightarrow$ sok írás
- REDO hátránya: Nem írunk, amíg nincs COMMIT $\Longrightarrow$ nagy memóriaigény

UNDO/REDO

Fő elv:

- Mielőtt az adatbázis bármely X elemének értékét a lemezen módosítanánk, a (T, X, v, w) naplóbejegyzésnek a lemezre kell kerülnie.

Nagyobb szabadság, hogy mikor írjunk.

UNDO/REDO protokoll-naplózás

- UNDO hátránya: COMMIT után azonnal kiírjuk a lemezre az értékeket $\Longrightarrow$ sok írás
- REDO hátránya: Nem írunk, amíg nincs COMMIT $\Longrightarrow$ nagy memóriaigény

UNDO/REDO

Fő elv:

- Mielőtt az adatbázis bármely X elemének értékét a lemezen módosítanánk, a (T, X, v, w) naplóbejegyzésnek a lemezre kell kerülnie.

Nagyobb szabadság, hogy mikor írjunk.
Nagyobb méretű napló. $\Longrightarrow$ v, w nagyon nagy is lehet!

Példa

T	t	A_M	B_M	A_D	B_D	*Napló*
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8, 16)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 8, 16)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
						(T, COMMIT)
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

UNDO/REDO visszaállítás

- A legkorábbitól kezdve állítsuk vissza minden befejezett tranzakció hatását. (REDO)

UNDO/REDO visszaállítás

- A legkorábbitól kezdve állítsuk vissza minden befejezett tranzakció hatását. (REDO)
- A legutolsótól kezdve állítsuk tegyük semmissé minden be nem fejezett tranzakció hatását. (UNDO)

Példa

T	t	A_M	B_M	A_D	B_D	Napló
				8	8	(T, BEGIN)
LOCK(A)				8	8	
LOCK(B)				8	8	
READ(A, t)	8	8		8	8	
$t := t \cdot 2$	16	8		8	8	
WRITE(A, t)	16	16		8	8	(T, A, 8, 16)
READ(B, t)	8	16	8	8	8	
$t := t \cdot 2$	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	(T, B, 8, 16)
(FLUSH LOG)						
OUTPUT(A)	16	16	16	16	8	
						(T, COMMIT)
OUTPUT(B)	16	16	16	16	16	
UNLOCK(A)						
UNLOCK(B)						

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a (START CHECKPOINT ($T_1, \ldots, T_k$)) bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. (FULSH LOG)

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a (START CHECKPOINT ($T_1, \ldots, T_k$)) bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. (FULSH LOG)
3. Írjuk a lemezre az összes piszkos puffert

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a (START CHECKPOINT ($T_1, \ldots, T_k$)) bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. (FULSH LOG)
3. Írjuk a lemezre az összes piszkos puffert
4. (END CHECKPOINT)
5. (FULSH LOG)

CHECKPOINT képzés működés közben

1. Írjuk a naplóba a (START CHECKPOINT ($T_1, \ldots, T_k$)) bejegyzést, ahol T_i az összes éppen aktív tranzakció
2. (FULSH LOG)
3. Írjuk a lemezre az összes piszkos puffert
4. (END CHECKPOINT)
5. (FULSH LOG)

Mindenképp elég visszamenni legfeljebb az előző CHEKPOINT-ig (mint a REDO-nál).

Példa

(T_1, BEGIN)
$(T_1, A, 4, 5)$
(T_2, BEGIN)
(T_1, COMMIT)
$(T_2, B, 9, 10)$
(START CHECKPOINT (T_2))
$(T_2, C, 14, 15)$
(T_3, BEGIN)
$(T_3, D, 19, 20)$
(END CHECKPOINT)
(T_2, COMMIT)
(T_3, COMMIT)

Védekezés lemezhiba ellen

- **A naplót külön lemezen tartjuk**
- **Nem dobjuk el a napló CHECKPOINT előtti részét sem**
- **REDO vagy UNDO/REDO protokollt használunk**

Védekezés lemezhiba ellen

- **A naplót külön lemezen tartjuk**
- **Nem dobjuk el a napló CHECKPOINT előtti részét sem**
- **REDO vagy UNDO/REDO protokollt használunk**

Így elvileg a kezdeti adatbázis ismeretében vissza tudjuk állítani a legutolsó állapotot.

Védekezés lemezhiba ellen

- **A naplót külön lemezen tartjuk**
- **Nem dobjuk el a napló CHECKPOINT előtti részét sem**
- **REDO vagy UNDO/REDO protokollt használunk**

Így elvileg a kezdeti adatbázis ismeretében vissza tudjuk állítani a legutolsó állapotot.
De a napló egy idő után nagyobb lesz, mint az adatbázis.
$\Longrightarrow$ Időnként archiválunk

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Helyreállítás

1. Megkeressük a legutolsó teljes mentést (volt (END DUMP))

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Helyreállítás

1. Megkeressük a legutolsó teljes mentést (volt (END DUMP))
2. Módosítjuk az adatbázist a napló segítségével a CHECKPOINT-tól kezdve (ezért kell REDO vagy UNDO/REDO)

Archiválás működés közben

Ha leállítjuk a rendszert, nyugodtan lehet menteni.
Ha nem lehet leállítani $\Longrightarrow$

1. (START DUMP) a naplóba
2. Megfelelő CHECKPOINT kialakítása
3. Adatok mentése valamilyen sorrendben
4. Napló mentése
5. (END DUMP)

Helyreállítás

1. Megkeressük a legutolsó teljes mentést (volt (END DUMP))
2. Módosítjuk az adatbázist a napló segítségével a CHECKPOINT-tól kezdve (ezért kell REDO vagy UNDO/REDO)

Osztott adatbázisok

- Adatok vízszintes felosztása

Osztott adatbázisok

- Adatok vízszintes felosztása
  - Egy bank több fiókja, saját ügyfelek

Osztott adatbázisok

- Adatok vízszintes felosztása
  - Egy bank több fiókja, saját ügyfelek
  - Üzlethálózat boltjai, saját eladások

Osztott adatbázisok

- Adatok vízszintes felosztása
  - Egy bank több fiókja, saját ügyfelek
  - Üzlethálózat boltjai, saját eladások
  - Könyvár több fiókkal, saját katalógussal

Osztott adatbázisok

- Adatok vízszintes felosztása
  - Egy bank több fiókja, saját ügyfelek
  - Üzlethálózat boltjai, saját eladások
  - Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
  - Bankban $\Longrightarrow$ ügyfél adatok helyben, hitelkártya adatok a központban

Osztott adatbázisok

- Adatok vízszintes felosztása
  - Egy bank több fiókja, saját ügyfelek
  - Üzlethálózat boltjai, saját eladások
  - Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
  - Bankban $\Longrightarrow$ ügyfél adatok helyben, hitelkártya adatok a központban
  - Üzletláncban $\Longrightarrow$ eladások helyben, megrendelések a központban

Osztott adatbázisok

- Adatok vízszintes felosztása
  - Egy bank több fiókja, saját ügyfelek
  - Üzlethálózat boltjai, saját eladások
  - Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
  - Bankban $\Longrightarrow$ ügyfél adatok helyben, hitelkártya adatok a központban
  - Üzletláncban $\Longrightarrow$ eladások helyben, megrendelések a központban
- Adotok többszörözése
  - Párhuzamosítás miatt

Osztott adatbázisok

- Adatok vízszintes felosztása
  - Egy bank több fiókja, saját ügyfelek
  - Üzlethálózat boltjai, saját eladások
  - Könyvár több fiókkal, saját katalógussal
- Adatok függőleges felbontása
  - Bankban $\Longrightarrow$ ügyfél adatok helyben, hitelkártya adatok a központban
  - Üzletláncban $\Longrightarrow$ eladások helyben, megrendelések a központban
- Adotok többszörözése
  - Párhuzamosítás miatt
  - Kommunikáció csökkentése miatt $\Longrightarrow$ gyakran szükséges adatok mindenhol (címjegyzék, telefonkönyv, chase-elés)

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.
Mikor lesz kész az egész tranzakció?

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.
Mikor lesz kész az egész tranzakció? $\Longrightarrow$ ha minden része kész

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.
Mikor lesz kész az egész tranzakció? $\Longrightarrow$ ha minden része kész
Hogyan vesszük észre? Mi van ha közben ABORT vagy hiba van?

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.
Mikor lesz kész az egész tranzakció? $\Longrightarrow$ ha minden része kész
Hogyan vesszük észre? Mi van ha közben ABORT vagy hiba van?

Példa

Áruházlánc központja lekérdezi minden boltban a mobiltelefon készletet. Ha valahol túl sok van, átküld belőle oda, ahol kevés van.

Osztott tranzakciók

A tranzakciók műveletei most különböző helyeken történhetnek.
Mikor lesz kész az egész tranzakció? $\Longrightarrow$ ha minden része kész
Hogyan vesszük észre? Mi van ha közben ABORT vagy hiba van?

Példa

Áruházlánc központja lekérdezi minden boltban a mobiltelefon készletet. Ha valahol túl sok van, átküld belőle oda, ahol kevés van.
$\Longrightarrow$ megszakadhat a kapcsolat menet közben, rossz az algoritmus, stb.

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - ★ Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - ★ Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - ★ Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - ★ Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - ★ Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - ★ Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - ★ Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$
    - (T, COMMIT) a saját naplójába

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$
    - (T, COMMIT) a saját naplójába
    - Mindenhova elküldi a (T, COMMIT-Lesz)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$
    - (T, COMMIT) a saját naplójába
    - Mindenhova elküldi a (T, COMMIT-Lesz)-t
  - Ha a koordinátor a (T, ABORT-Legyen)-t kapta legalább egy állomástól

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$
    - (T, COMMIT) a saját naplójába
    - Mindenhova elküldi a (T, COMMIT-Lesz)-t
  - Ha a koordinátor a (T, ABORT-Legyen)-t kapta legalább egy állomástól
    - (T, ABORT) a saját naplójába

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$
    - (T, COMMIT) a saját naplójába
    - Mindenhova elküldi a (T, COMMIT-Lesz)-t
  - Ha a koordinátor a (T, ABORT-Legyen)-t kapta legalább egy állomástól
    - (T, ABORT) a saját naplójába
    - Mindenhova elküldi a (T, ABORT-Lesz)-t

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$
    - (T, COMMIT) a saját naplójába
    - Mindenhova elküldi a (T, COMMIT-Lesz)-t
  - Ha a koordinátor a (T, ABORT-Legyen)-t kapta legalább egy állomástól
    - (T, ABORT) a saját naplójába
    - Mindenhova elküldi a (T, ABORT-Lesz)-t
  - Ha egy állomás a (T, COMMIT-Lesz)-t kapja $\Longrightarrow$ (T, COMMIT)

Kétfázisú véglegesítés (2PC)

- Alapelvek
  - Minden állomás naplózza saját eseményeit
  - Van egy koordinátor állomás, aki a döntést hozza majd
  - Az állomások üzeneteket küldenek egymásnak, ezeket is naplózzák (ki- és bejövőt is)
- Első fázis
  - A koordinátor saját naplójába (T, Felkészül)
  - Ezt mindenhova elküldi (még magának is)
  - Ha egy állomás megkapta az üzenetet, eldönti, hogy a nála található részre COMMIT vagy ABORT lesz majd
    - Ha COMMIT várható (már csak ez lenne hátra) $\Longrightarrow$ (T, Készenáll) a saját naplóba
      A koordinátornak elküldeni (T, Készenáll)-t
    - Ha ABORT várható $\Longrightarrow$ (T, ABORT-Legyen) a saját naplóba
      A koordinátornak elküldeni (T, ABORT-Legyen)-t
- Második fázis
  - Ha a koordinátor a (T, Készenáll)-t megkapta mindenkitől $\Longrightarrow$
    - (T, COMMIT) a saját naplójába
    - Mindenhova elküldi a (T, COMMIT-Lesz)-t
  - Ha a koordinátor a (T, ABORT-Legyen)-t kapta legalább egy állomástól
    - (T, ABORT) a saját naplójába
    - Mindenhova elküldi a (T, ABORT-Lesz)-t
  - Ha egy állomás a (T, COMMIT-Lesz)-t kapja $\Longrightarrow$ (T, COMMIT)
  - Ha egy állomás a (T, ABORT-Lesz)-t kapja $\Longrightarrow$ (T, ABORT)

Helyreállítás

Egy adott állomáson:

- Ha az utolsó bejegyzés COMMIT, ABORT, COMMIT-Lesz vagy ABORT-Lesz akkor a napló szerint helyreállítunk

Helyreállítás

Egy adott állomáson:

- Ha az utolsó bejegyzés COMMIT, ABORT, COMMIT-Lesz vagy ABORT-Lesz akkor a napló szerint helyreállítunk
- Ha az utolsó bejegyzés Készenáll, akkor nem világos a helyzet, vagy várunk, vagy kommunikálunk a többivel, vagy . . .

Helyreállítás

Egy adott állomáson:

- Ha az utolsó bejegyzés COMMIT, ABORT, COMMIT-Lesz vagy ABORT-Lesz akkor a napló szerint helyreállítunk
- Ha az utolsó bejegyzés Készenáll, akkor nem világos a helyzet, vagy várunk, vagy kommunikálunk a többivel, vagy . . .
- Ha nincs semmilyen bejegyzés, akkor ABORT (vagy várunk)

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat $\Longrightarrow$ globális (logikai) LOCK és lokális LOCK

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat $\Longrightarrow$ globális (logikai) LOCK és lokális LOCK
Egyszerű modell
Minden LOCK globális és az egyik állomás nyilvántartja ezeket

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat $\Longrightarrow$ globális (logikai) LOCK és lokális LOCK
Egyszerű modell
Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat $\Longrightarrow$ globális (logikai) LOCK és lokális LOCK
Egyszerű modell
Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás
Költség: egy LOCK-hoz 3 üzenet $\Longrightarrow$ igénylés, engedélyezés, feloldás

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat $\Longrightarrow$ globális (logikai) LOCK és lokális LOCK
Egyszerű modell
Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás
Költség: egy LOCK-hoz 3 üzenet $\Longrightarrow$ igénylés, engedélyezés, feloldás
$\Longrightarrow$ a zárállomás nagyon leterhelt lehet

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat $\Longrightarrow$ globális (logikai) LOCK és lokális LOCK
Egyszerű modell
Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás
Költség: egy LOCK-hoz 3 üzenet $\Longrightarrow$ igénylés, engedélyezés, feloldás
$\Longrightarrow$ a zárállomás nagyon leterhelt lehet
Elsődleges példány
Van egy elsődleges példány, ha valaki zárolni akar valamit, akkor az elsődleges példányt tároló állomáshoz fordul.

Osztott zárolás

Ha nincs adattöbbszörözés $\Longrightarrow$ $\surd$
Ha van $\Longrightarrow$ összhangban kell tartani a példányokat $\Longrightarrow$ globális (logikai) LOCK és lokális LOCK
Egyszerű modell
Minden LOCK globális és az egyik állomás nyilvántartja ezeket zárállomás
Költség: egy LOCK-hoz 3 üzenet $\Longrightarrow$ igénylés, engedélyezés, feloldás
$\Longrightarrow$ a zárállomás nagyon leterhelt lehet
Elsődleges példány
Van egy elsődleges példány, ha valaki zárolni akar valamit, akkor az elsődleges példányt tároló állomáshoz fordul.
Költség: mint előbb, de nem koncentrált forgalom

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK *A*-ja

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK *A*-ja
- Ha egy tranzakciónak van globális WLOCK *A*-ja, akkor egy másiknak nem lehet globális RLOCK *A*-ja

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK *A*-ja
- Ha egy tranzakciónak van globális WLOCK *A*-ja, akkor egy másiknak nem lehet globális RLOCK *A*-ja
- Lehet több tranzakciónak globális RLOCK *A*-ja

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK *A*-ja
- Ha egy tranzakciónak van globális WLOCK *A*-ja, akkor egy másiknak nem lehet globális RLOCK *A*-ja
- Lehet több tranzakciónak globális RLOCK *A*-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK *A*-ja
- Ha egy tranzakciónak van globális WLOCK *A*-ja, akkor egy másiknak nem lehet globális RLOCK *A*-ja
- Lehet több tranzakciónak globális RLOCK *A*-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Hogyan lehet megszerezni egy globális RLOCK-ot vagy WLOCK-ot?

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK *A*-ja
- Ha egy tranzakciónak van globális WLOCK *A*-ja, akkor egy másiknak nem lehet globális RLOCK *A*-ja
- Lehet több tranzakciónak globális RLOCK *A*-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Hogyan lehet megszerezni egy globális RLOCK-ot vagy WLOCK-ot? $\Longrightarrow$ többféle modell

Osztott RLOCK/WLOCK

Alapelvek:

- Semelyik két tranzakciónak nem lehet globális WLOCK *A*-ja
- Ha egy tranzakciónak van globális WLOCK *A*-ja, akkor egy másiknak nem lehet globális RLOCK *A*-ja
- Lehet több tranzakciónak globális RLOCK *A*-ja
- Minden állomás az érvényes globális lock-ok figyelembevételével működik

Hogyan lehet megszerezni egy globális RLOCK-ot vagy WLOCK-ot? $\Longrightarrow$ többféle modell

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi
  - Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i-t $\Longrightarrow$ globális RLOCK A

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi
  - Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i-t $\Longrightarrow$ globális RLOCK A
- Globális WLOCK A megszerzése
  - Ha az i állomás akar egy WLOCK A_i-t, akkor üzen minden másik helyre ahol van A_j

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi
  - Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i-t $\Longrightarrow$ globális RLOCK A
- Globális WLOCK A megszerzése
  - Ha az i állomás akar egy WLOCK A_i-t, akkor üzen minden másik helyre ahol van A_j
  - Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i
- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi
  - Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i-t $\Longrightarrow$ globális RLOCK A
- Globális WLOCK A megszerzése
  - Ha az i állomás akar egy WLOCK A_i-t, akkor üzen minden másik helyre ahol van A_j
  - Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
  - Ha mindenhonnan engedélyezés jött, az i állomás felteszi a WLOCK A_i-t, mindenhova üzen, hogy WLOCK A_j-t $\Longrightarrow$ globális WLOCK A

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i

- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi
  - Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i-t $\Longrightarrow$ globális RLOCK A
- Globális WLOCK A megszerzése
  - Ha az i állomás akar egy WLOCK A_i-t, akkor üzen minden másik helyre ahol van A_j
  - Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
  - Ha mindenhonnan engedélyezés jött, az i állomás felteszi a WLOCK A_i-t, mindenhova üzen, hogy WLOCK A_j-t $\Longrightarrow$ globális WLOCK A

$\Longrightarrow$ Ha az egyik állomás kért és kapott WLOCK A-t, akkor másik nyilván nem kaphat később sem WLOCK A-t, sem RLOCK A-t

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i

- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi
  - Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i-t $\Longrightarrow$ globális RLOCK A
- Globális WLOCK A megszerzése
  - Ha az i állomás akar egy WLOCK A_i-t, akkor üzen minden másik helyre ahol van A_j
  - Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
  - Ha mindenhonnan engedélyezés jött, az i állomás felteszi a WLOCK A_i-t, mindenhova üzen, hogy WLOCK A_j-t $\Longrightarrow$ globális WLOCK A

$\Longrightarrow$ Ha az egyik állomás kért és kapott WLOCK A-t, akkor másik nyilván nem kaphat később sem WLOCK A-t, sem RLOCK A-t

$\Longrightarrow$ Ha az egyik állomás kért és kapott RLOCK A-t, akkor másik nyilván nem kaphat később WLOCK A-t, de kaphat RLOCK A-t

WALL (write locks all)

- Globális RLOCK A megszerzéséhez elég egy lokális RLOCK A_i
- Globális WLOCK A megszerzéséhez kell minden lokális WLOCK A_i

- Globális RLOCK A megszerzése:
  - Ha az i állomás akar egy RLOCK A_i-t, nem kell üzenni, megnézzük milyen zár van A_i-n
  - Ha itt WLOCK A_i van, akkor elutasítja a kérést, ha semmi vagy RLOCK A_i, akkor engedélyezi
  - Ha engedélyezi, akkor az i állomás felteszi az RLOCK A_i-t $\Longrightarrow$ globális RLOCK A
- Globális WLOCK A megszerzése
  - Ha az i állomás akar egy WLOCK A_i-t, akkor üzen minden másik helyre ahol van A_j
  - Ha itt RLOCK A_j vagy WLOCK A_j van, akkor elutasítja a kérést, ha semmi akkor engedélyezi
  - Ha mindenhonnan engedélyezés jött, az i állomás felteszi a WLOCK A_i-t, mindenhova üzen, hogy WLOCK A_j-t $\Longrightarrow$ globális WLOCK A

$\Longrightarrow$ Ha az egyik állomás kért és kapott WLOCK A-t, akkor másik nyilván nem kaphat később sem WLOCK A-t, sem RLOCK A-t

$\Longrightarrow$ Ha az egyik állomás kért és kapott RLOCK A-t, akkor másik nyilván nem kaphat később WLOCK A-t, de kaphat RLOCK A-t

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i-k többségén
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen az A_i-k többségén

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i-k többségén
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen az A_i-k többségén

$\Longrightarrow$ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i-k többségén
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen az A_i-k többségén

$\Longrightarrow$ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Miért jó a többségi zárolás?

Két tranzakció nem tud egyszerre WLOCK A-t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i-k többségén
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen az A_i-k többségén

$\Longrightarrow$ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.
Miért jó a többségi zárolás?
Két tranzakció nem tud egyszerre WLOCK A-t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia $\Longrightarrow$ van olyan példány, amire mindkettő kapna

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i-k többségén
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen az A_i-k többségén

$\Longrightarrow$ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.

Miért jó a többségi zárolás?

Két tranzakció nem tud egyszerre WLOCK A-t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia $\Longrightarrow$ van olyan példány, amire mindkettő kapna

Hasonlóan nem lehet egy tranzakcióknak WLOCK A-ja, egy másiknak RLOCK A-ja.

Többségi zárolás

Csak az különbözik, hogy hogyan lehet globális zárat szerezni:

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen az A_i-k többségén
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen az A_i-k többségén

$\Longrightarrow$ Több üzenet szükséges az RLOCK megszerzéséhez, de kevesebb a WLOCK-hoz, mint az előbb.
Miért jó a többségi zárolás?
Két tranzakció nem tud egyszerre WLOCK A-t szerezni, mert mindkettőnek a példányok több, mint felére kellene WLOCK-ot kapnia $\Longrightarrow$ van olyan példány, amire mindkettő kapna
Hasonlóan nem lehet egy tranzakcióknak WLOCK A-ja, egy másiknak RLOCK A-ja.
Viszont lehet két különböző tranzakciónak RLOCK A-ja, hiszen egy példányon is lehet ilyen.

k az n-ből protokoll

Közös általánosítás:
Legyen n, hogy hány példány van A-ból és legyen $n \geq k \geq \lceil (n+1)/2 \rceil$

k az n-ből protokoll

Közös általánosítás:
Legyen n, hogy hány példány van A-ból és legyen $n \geq k \geq \lceil (n+1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n+1-k$ db A_i-n
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen legalább k db A_i-n

k az n-ből protokoll

Közös általánosítás:
Legyen n, hogy hány példány van A-ból és legyen $n \geq k \geq \lceil (n+1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n+1-k$ db A_i-n
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen legalább k db A_i-n

$k = n \implies$ WALL
$k = \lceil (n+1)/2 \rceil \implies$ többségi zárolás

k az n-ből protokoll

Közös általánosítás:
Legyen n, hogy hány példány van A-ból és legyen $n \geq k \geq \lceil (n+1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n+1-k$ db A_i-n
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen legalább k db A_i-n

$k = n \implies$ WALL
$k = \lceil (n+1)/2 \rceil \implies$ többségi zárolás
k választásával hangolható a költség.
Miért jó a protokoll?

k az n-ből protokoll

Közös általánosítás:
Legyen n, hogy hány példány van A-ból és legyen $n \geq k \geq \lceil (n+1)/2 \rceil$

- Globális RLOCK A megszerzéséhez kell, hogy lokális RLOCK A_i legyen legalább $n+1-k$ db A_i-n
- Globális WLOCK A megszerzéséhez kell , hogy lokális WLOCK A_i legyen legalább k db A_i-n

$k = n \implies$ WALL
$k = \lceil (n+1)/2 \rceil \implies$ többségi zárolás
k választásával hangolható a költség.
Miért jó a protokoll? $\implies$ hasonlóan a többségi bizonyításához

Adatbázisok elmélete

Feladatmegoldások tranzakciókezeléshez

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

27. előadás

Feladat

- Tegyük fel, hogy az alábbi műveletsorozatban minden egyes olvasás- és írásműveletet közvetlenül megelőzi az RLOCK ill. a WLOCK igénylése. Tegyük továbbá fel, hogy a zárak feloldása a tranzakció utolsó művelete után történik meg. Adjuk meg azokat a műveleteket, melyek végrehajtását az ütemező megtagadja, és mondjuk meg, hogy létrejön-e holtpont. Hogyan alakul a műveletek végrehajtása során a várakozási gráf? Ha létrejön holtpont, ABORT-áljuk az egyik tranzakciót, és mutassuk meg, hogyan folytatódik a műveletsorozat!

$$r_1(A), r_2(B), w_1(C), r_3(D), r_4(E), w_3(B), w_2(C), w_4(A), w_1(D)$$

Megoldás

Először sorban kérünk zárakat A, B, C, D, E-re.

$$rl_1(A), r_1(A), rl_2(B), r_2(B), wl_1(C), w_1(C), rl_3(D), r_3(D), rl_4(E)$$

Az első probléma a $wl_3(B)$, hiszen ekkor van zár még $lr_2(B)$. $wl_3(B)$ megtagadva, várakozási gráfba (T_3, T_2) él, T_3 vár
$wl_2(C)$ megtagadva, várakozási gráfba (T_2, T_1) él, T_2 vár
$wl_4(A)$ megtagadva, várakozási gráfba (T_4, T_1) él, T_4 vár
$wl_1(D)$ megtagadva, várakozási gráfba (T_1, T_3) él, kört kapunk, holtpont alakul ki.
ABORT T_1, ekkor eltűnik (T_2, T_1) él és a (T_4, T_1) él a várakozási gráfból, ami így DAG lesz. Ezért pl. T_2, T_3, T_4 sorrendben lefuthat a többi tranzakció:

$$ul_1(A), ul_1(C), wl_2(C), w_2(C), ul_2(C), ul_2(B), wl_3(B),$$

$$w_3(B), ul_3(B), wl_1(A), w_4(A), ul_4(A), ul_4(E)$$

Feladat

- Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A), \quad w_3(B), \quad r_1(A), \quad w_2(B), \quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)
Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és
(a) ne kövesse mindegyik tranzakció a 2PL-t, de (a zárkérések alapján döntve) az ütemezés sorosítható legyen,

Megoldás

$$
\begin{aligned}
&l_2(A),\ r_2(A),\ u_2(A),\\
&l_3(B),\ w_3(B),\ u_3(B),\\
&l_1(A),\ r_1(A),\ u_1(A),\\
&l_2(B),\ w_2(B),\ u_2(B),\\
&l_1(C),\ w_1(C),\ u_1(C)
\end{aligned}
$$

Ha felrajzoljuk a sorosítási gráfot: $T_3 \to T_2 \to T_1$, tehát sorosítható.

Feladat

- Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A), \quad w_3(B), \quad r_1(A), \quad w_2(B), \quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)
Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és
(b) mindegyik tranzakció kövesse a 2PL-t, de (a zárkérések alapján döntve) ne legyen sorosítható az ütemezés,

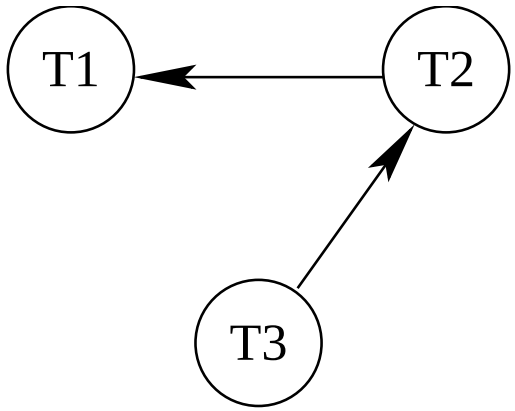
Megoldás:
Ilyet nem lehet adni, mert tanultuk azt a tételt, hogy ha minden tranzakció követi a 2PL-t, akkor sorosítható lesz az ütemezés.

Feladat

- (c) mindegyik tranzakció kövesse a 2PL-t, és (a zárkérések alapján döntve) legyen sorosítható az ütemezés.

Megoldás:
Az ötlet az, hogy az $l_2(B)$-t előbbre lehet hozni és így előbb fel lehet oldani a zárat A-n.

$$l_2(A),\ r_2(A),$$
$$l_3(B),\ w_3(B),\ u_3(B),$$
$$l_2(B),\ u_2(A),\ l_1(A),\ r_1(A),$$
$$w_2(B),\ u_2(B),$$
$$l_1(C),\ w_1(C),\ u_1(A),\ u_1(C)$$



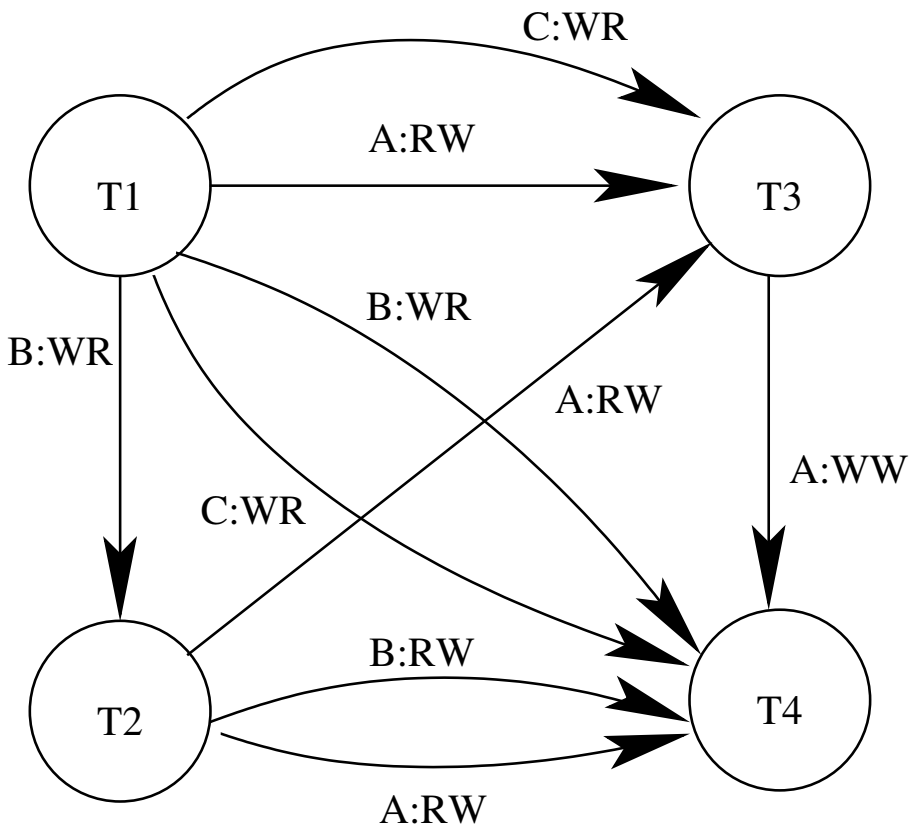
(Mivel ez 2PL, a tétel szerint sorosítható.)

Feladat

- Az alábbi legális ütemezés négy tranzakció zárjait tartalmazza az RLOCK/WLOCK modellben. Rajzoljuk fel a sorosítási gráfot! Sorosítható-e az ütemezés? Ha igen, milyen soros ütemezések ekvivalensek az eredeti ütemezéssel?

(0)	T_1	T_2	T_3	T_4
(1)		RLOCK A		
(2)	RLOCK A			
(3)	WLOCK C			
(4)	UNLOCK C			
(5)			RLOCK C	
(6)	WLOCK B			
(7)	UNLOCK B			
(8)				RLOCK B
(9)	UNLOCK A			
(10)		UNLOCK A		
(11)			WLOCK A	
(12)				RLOCK C
(13)		WLOCK D		
(14)				UNLOCK B
(15)			UNLOCK C	
(16)		RLOCK B		
(17)			UNLOCK A	
(18)				WLOCK A
(19)		UNLOCK B		
(20)				WLOCK B
(21)				UNLOCK B
(22)		UNLOCK D		
(23)				UNLOCK C
(24)				UNLOCK A

Megoldás:



Sorosítható, mert DAG. Egy ekvivalens soros ütemezés van: T_1, T_2, T_3, T_4

Feladat

- Az alább megadott tranzakciók mindegyikénél tételezzük fel, hogy beszúrjuk a LOCK és UNLOCK műveletet miden egyes adatbáziselemhez, amihez hozzáférünk: $r_1(A)$, $w_1(B)$.
  Adjuk meg, hogy a zárolási, feloldási, olvasási és írási műveleteknek hány olyan sorrendje lehet, ha a zárolások megfelelőek és a zárolás i) kétfázisú, ii) nem kétfázisú.

Megoldás:

Ha a zárolás megfelelő, akkor csak ennek összefésülései jönnek szóba:
a) $l_1(A); r_1(A); u_1(A)$ b) $l_1(B); w_1(B); u_1(B)$
ii) csak akkor nem kétfázisú, ha az egyik megelőzi a másikat: kétféle ilyen van.
i) Hány összefésülés van összesen? Ismétléses kombináció, vagy a 6 pozícióból melyik 3 az elsőből: $\frac{6!}{3!\cdot 3!} = \binom{6}{3} = 20$, tehát i)=18.

Feladat

- Az alábbi legális ütemezés két olyan tranzakció utasításait tartalmazza, melyek betartják a figyelmeztető protokollt. Hogy nézhet ki az ütemezésben szereplő adategységek egymásba ágyazottságát reprezentáló fa, ha tudjuk, hogy a gyökérnek legfeljebb 3 gyereke van? (Ha több lehetséges eset van, akkor mindet add meg).
  $WARN_1(E)$, $WARN_1(H)$, $WARN_2(E)$, $LOCK_1(A)$, $LOCK_1(C)$, $UNLOCK_1(A)$, $LOCK_2(F)$, $UNLOCK_1(H)$, $UNLOCK_2(F)$, $UNLOCK_1(C)$, $UNLOCK_2(E)$, $UNLOCK_1(E)$

Megoldás:

T_2 miatt biztos, hogy E a gyökér és F ennek a fia.
T_1 miatt biztos, hogy H is E-nek a fia.
A és C helyzete a kérdéses még. Két eset lehetséges:

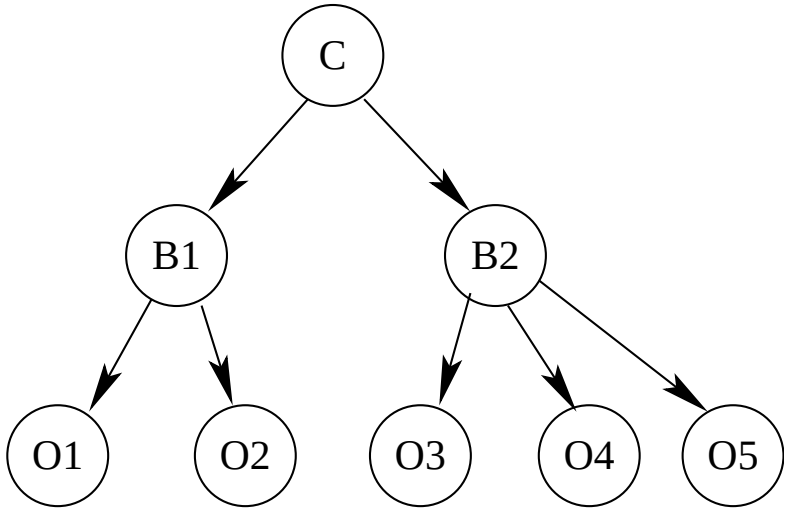
- Az A csúcs a H gyereke: C nem lehet se A, se H gyereke, mert később oldjuk fel C-n a zárat, mint A-n és H-n, így ekkor C csak E gyereke lehet és ez összhangban is van a zárolással. Ez egy lehetséges megoldás.
- Az A csúcs az E gyereke: C nem lehet se A, se H gyereke a zárfeloldások miatt, de E-é se lehet, mert a gyökérnek csak három gyereke lehet. Így ezen az ágon nem kapunk megoldást.

Feladat

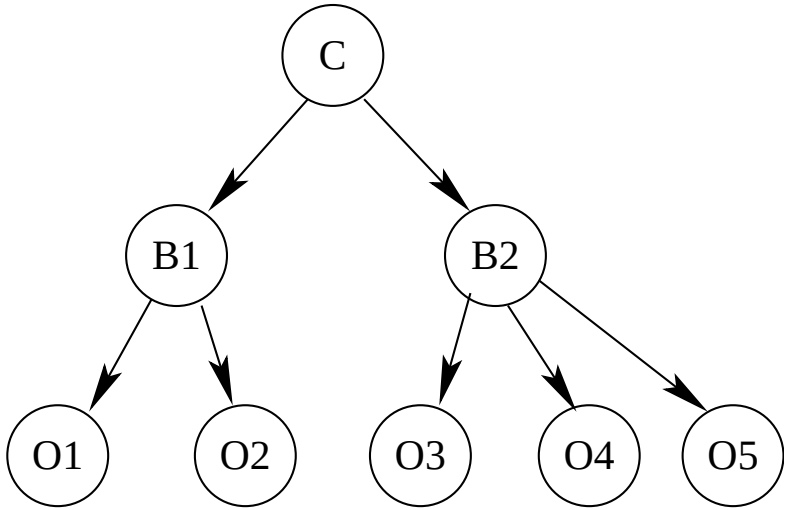
- Vegyünk egy objektum orientált adatbázist. A C osztály objektumait két blokkban tároljuk a B_1-ben és a B_2-ben. A B_1 tartalmazza az O_1 és O_2 objektumokat, míg a B_2 az O_3, O_4, O_5 objektumokat. Adjuk meg a zárolási kérések sorozatát és a figyelmeztető protokoll alapú ütemező feladatát az alábbi kérési sorozatokhoz. Feltehetjük, hogy minden kérés éppen azelőtt fordul elő, mint amikor éppen szükség van rá, és minden zárfeloldás a tranzakció befejeztével történik. Használjuk az RLOCK/WLOCK modellt.

$$r_1(O_1), w_2(O_2), r_2(O_3), w_1(O_4)$$

Megoldás: A hierarchiát ábrázoló fa:



Megoldás: A hierarchiát ábrázoló fa:



- Az első körben O_1-re kell majd zárat tenni:
  $RWARN_1(C)$, $RWARN_1(B_1)$, $RLOCK_1(O_1)$.
- A második körben O_2-re kell majd zárat tenni:
  $WWARN_2(C)$, $WWARN_2(B_1)$, $WLOCK_2(O_2)$.
- A harmadik körben O_3-ra kell majd zárat tenni:
  $RWARN_2(C)$, $RWARN_2(B_2)$, $RLOCK_2(O_3)$. Ezután T_2 felengedi a zárakat és figyelmeztetéseit:
  $UNLOCK_2(O_3)$, $UNLOCK_2(B_2)$, $UNLOCK_2(O_2)$, $UNLOCK_2(B_1)$, $UNLOCK_2(C)$
- A negyedik körben O_4-re kell majd zárat tenni:
  $WWARN_1(C)$, $WWARN_1(B_2)$, $WLOCK_1(O_4)$. Ezután T_1 felengedi a zárakat és figyelmeztetéseit:
  $UNLOCK_1(O_4)$, $UNLOCK_1(B_4)$, $UNLOCK_1(O_1)$, $UNLOCK_1(B_1)$, $UNLOCK_1(C)$

Feladat

- Tekintsük a T_1, T_2, T_3 és T_4 tranzakciók írási és olvasási kéréseiből álló

$$r_1(A),\ w_2(B),\ r_3(A),\ w_1(B),\ r_2(A),\ r_4(B),\ w_4(A),\ w_3(B)$$

sorozatot. Időbélyeges tranzakciókezeléssel akarjuk a sorosítható ütemezést kikényszeríteni, a tranzakciók időbélyegei: $t(T_1) = 1$, $t(T_2) = 2$, $t(T_3) = 3$, $t(T_4) = 4$. Írja le, hogy mi történik a fenti sorozat esetén, azaz mely kéréseket teljesíti az ütemező, melyeket nem, mely kérések vezetnek ABORT-hoz és adja meg azt is, hogy hogyan változnak az egyes műveletek után az adategységek írási és olvasási idejei (ezek kezdetben mind nullák).

Megoldás:

Kérés	r(A)	w(A)	r(B)	w(B)	Magyarázat
	0	0	0	0	kezdetben minden nulla
$r_1(A)$	1	0	0	0	mehet
$w_2(B)$	1	0	0	2	mehet
$r_3(A)$	3	0	0	2	mehet
$w_1(B)$	3	0	0	2	Nem lesz ABORT, de írás sem
$r_2(A)$	3	0	0	2	Nem lesz ABORT, $r(A)$ nem változik
$r_4(B)$	3	0	4	2	mehet
$w_4(A)$	3	4	4	2	mehet
$w_3(B)$	3	4	4	2	ABORT

Feladat

- Alább látható egy napló, melyet az UNDO/REDO protokoll szerint képeztünk. Állítsuk vissza a konzisztens helyzetet.

(T_1, BEGIN)
$(T_1, A, 4, 5)$
(T_2, BEGIN)
(T_1, COMMIT)
$(T_2, B, 9, 10)$
(START CHECKPOINT (T_2))
$(T_2, C, 14, 15)$
(T_3, BEGIN)
$(T_3, D, 19, 20)$
$(T_3, A, 5, 6)$
$(T_3, E, 10, 15)$
(T_4, BEGIN)
$(T_4, F, 15, 16)$
(END CHECKPOINT)
(T_2, COMMIT)
HIBA

Megoldás:
Mivel van END CHECKPOINT, elég a START checkpoint-tól előre haladva megállapítjuk, hogy befejeződött T_2, nem fejeződött be T_3 és T_4. (T_1-el nem kell foglalkozni, mert már minden adata kiíródott.)
A T_2 elejéről indulva $B := 10$, de ez már nem kell, hiszen lemezen van. Majd $C := 15$.
Utána végéről visszafelé $F := 15, E := 10, A := 5, D := 19$, ABORT T_3 és ABORT T_4, FLUSH LOG