10.előadás: Adatbázisok-I.

dr. Hajas Csilla (ELTE IK)

http://sila.hajas.elte.hu/

SQL – PL/SQL témakör befejező része

1.fej. Adatbázis-kezelő rendszerek áttekintése

[7.1.-7.3. Táblák és megszorítások -- volt: 7.ea]

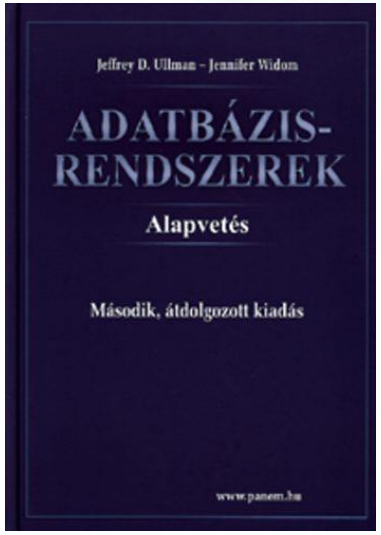
7.4.-7.5. folyt.: Önálló megszorítások, triggerek

8.1.-8.2. Nézettáblák, tárolt nézettáblák

[9.3.-9.4. SQL/PSM, Oracle PL/SQL --volt: 8.ea]

10.1. Jogosultságok; // és 10.2. Rekurzió

http://sila.hajas.elte.hu/AB1ea/SQL5_adatb_hcs.pdf



Eddigi előadásokról rövid áttekintés

-- Egy táblára vonatkozó ismeretek

- [01] TERV1.pdf (Relációs modell és az E/K modell bev)
  SQL1.pdf (SQL bev, create table/1.tipusok, kulcsok)
- [02] REL1.pdf (Egytáblás lekérdezések, vetítés, szűrés)
- [03] REL2.pdf (Egytáblás lekérdezések, csoportosítás)

-- Több táblára vonatkozó ismeretek

- [04] TERV2.pdf (E/K haladó, megszorítások, alosztályok)
  SQL2.pdf (create table/2., megszorítások, hivatk.épség)
- [05] REL3.pdf (Több táblás lekérd. relációs algebrában)
- [06] REL4.pdf (Több táblás lekérdezések az SQL-ben)

-- SQL a gyakorlatban

- [07] SQL3.pdf SQL/DML insert, delete, update, SQL/DDL …
- [08] SQL4.pdf SQL/PSM, sémában tárolt eljárások, PL/SQL
- [09] REL5.pdf Befejező rész: Relációs lekérdezések/Datalog
- [10] SQL5.pdf Befejező rész: SQL a gyakorlatban témakör

Adatbázisok-1 (vizsgatematika)

- Szorgalmi időszak: Előadás+gyakorlat (gépes labor gyak: SQL) az előadás anyag nem lineáris, hanem a gyakorlatot támogató!

Vizsgák: az előadások három témaköre vizsgán is három témakör:

- REL [Relációs lekérdezések] SQL, mint lekérdező nyelv, SELECT utasítás és végrehajtása; relációs algebra; logika
- SQL [SQL és PL/SQL] Adatbázis-kezelő rendszer felépítése, SQL, mint adatbázis-kezelő nyelv, fő részei, SQL DDL, DML; SQL programnyelvi környezetben: Oracle PL/SQL; Rekurzió
- TERV [Tervezés] Relációs adatmodell, Egyed/Kapcs.modell, E/K diagram átalakítása relációs sémákra, megszorítások, relációs sématervezés, függőségek elmélete, normalizálás

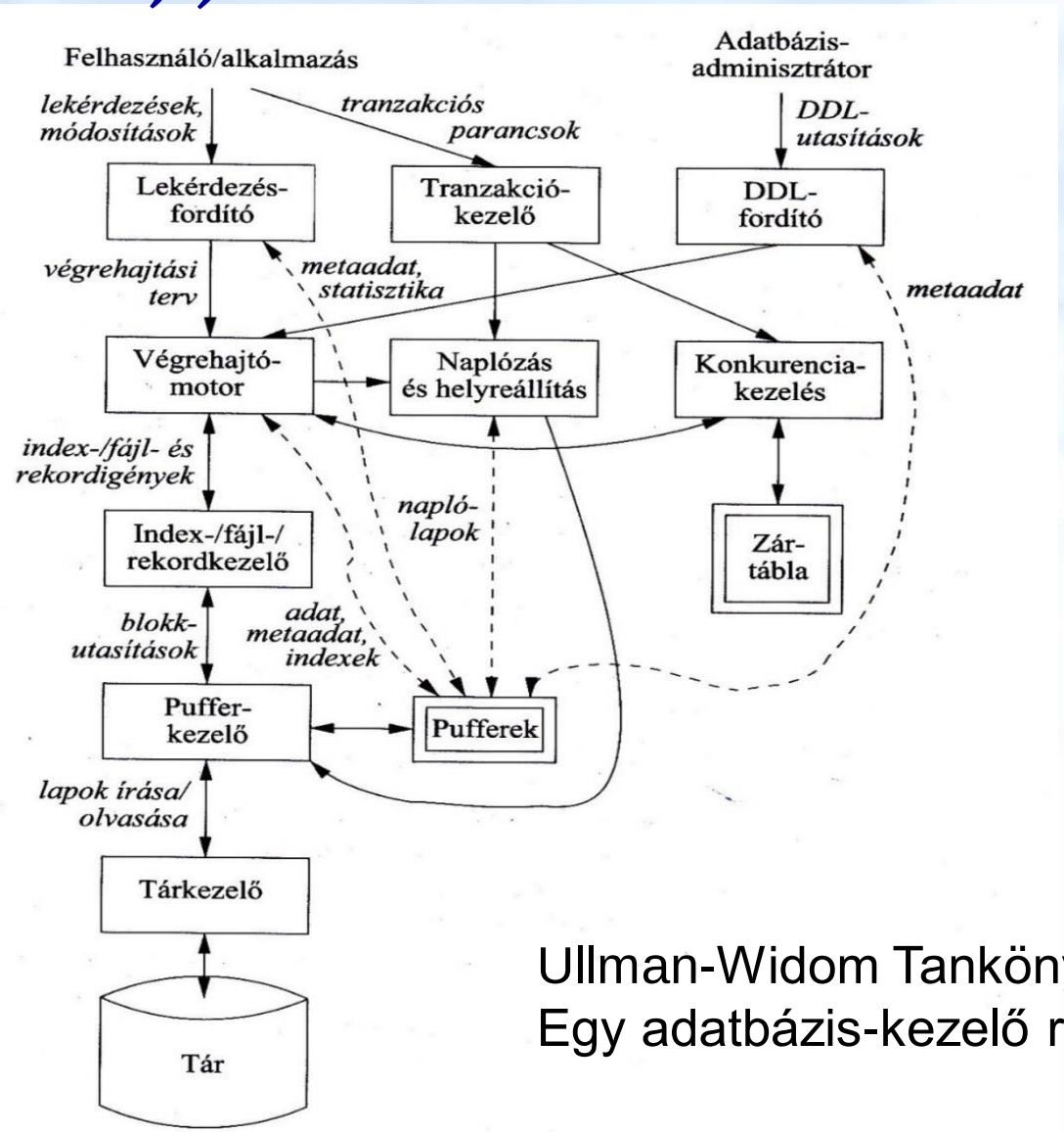
A vizsgaidőszak minden hetében hetente egy vizsgaalkalom

- Kedd 12:00-14:00 online írásbeli vizsga a Canvasban: A három témakörből 3-3 feladat megoldásának beküldése

Adatbázisok-1 (Canvas-vizsga)

- Ebben a félévben az online oktatás miatt több módszertani változást vezettünk be. A korábbiakban megszokott "papíros" feladatok helyett, ahol csak lehet számítógépen lefuttatandó feladatokat, programokat kérünk számon. Így lesz ez majd a vizsgán is, ezért még a vizsgaidőszak előtt a gyakorlatokon szeretnénk ha megismernék és kipróbálnák azokat az eszközöket, amelyeket majd a vizsgán használniuk kell.
- A relációs algebrai lekérdezéseket a korábbiakban "papíron" kell(ett) megfogalmazniuk, ismerjenek meg egy olyan környezetet, amelynek segítségével le is tudják futtatni a relációs algebrában megfogalmazott lekérdezéseket. A vizsgán ezt a környezetet is használniuk kell majd, ezért kérem, hogy gyakorolják az eszköz használatát.
- A környezet elérése: https://dbis-uibk.github.io/relax/calc.htm
  Kérem olvassák el a használatra vonatkozó Help-et.

(Tk.1.fej.) Az adatbázis-kezelő rendszerek áttekintése



Ullman-Widom Tankönyv 1.1. ábra
Egy adatbázis-kezelő rendszer részei

Mit várunk egy ABKR-től? --- 1

- Visszatérünk a Tankönyv 1.fejezetére
- Adatbázis: Adatok együttese, amelyet az adatbázis-kezelő rendszer kezel. Mit várunk az ABKR-től? (DBMS-től?)
  - Tegye lehetővé a felhasználók számára, hogy új adatbázisokat hozhassanak létre, és azok sémáját, vagyis az adatok logikai struktúráját egy speciális nyelven adhassák meg: Adatdefiníciós nyelv (DDL)
  - Tegye lehetővé a felhasználóknak, hogy az adatokat egy megfelelő nyelv segítségével lekérdezhessék vagy módosíthassák: Adatkezelő nyelv (DML)
  - Kényelmes (fizikai adatfüggetlenség, magas szintű deklaratív nyelv, mint például az SQL szabvány)
  - Hatékony legyen a megvalósítás.

Mit várunk egy ABKR-től? --- 2

- (folyt.) Mit várunk az ABKR-től? (DBMS-től?)
  - Támogassa nagy méretű (több terabyte mennyiségű) adat hosszú időn keresztül való tárolását, és tegye lehetővé a hatékony hozzáférést a lekérdezések és adatbázis-módosítások számára.
  - Biztosítsa a tartósságot, az adatb. helyreállíthatóságát, biztonságos (konzisztens állapot biztosítsa, védve legyen a hardware, software és felhasználói hibáktól).
  - Felügyelje a több felhasználó által egy időben történő adathozzáféréseket úgy, hogy ezek a műveletek ne legyenek hatással a többi felhasználóra számára (konkurencia-vezérlés)
  - A fenti pontok részletesebben kifejtve:

(1) Adatbázis-kezelés

Adatbázis-kezelés:

- (1) Háttértárolón tárolt, nagy adatmennyiség hatékony kezelése (lekérdezése, módosítása)
- (2) Adatmodell támogatása
- (3) Adatbázis-kezelő nyelvek támogatása
- (4) Több felhasználó támogatása
- (5) Adatvédelem, adatbiztonság
- (6) Tranzakció-kezelés
- (7) Konkurencia-kezelés
- (8) Naplózás és helyreállíthatóság
- (9) Lekérdezések végrehajtásának optimalizálása

(2) Adatmodell támogatása

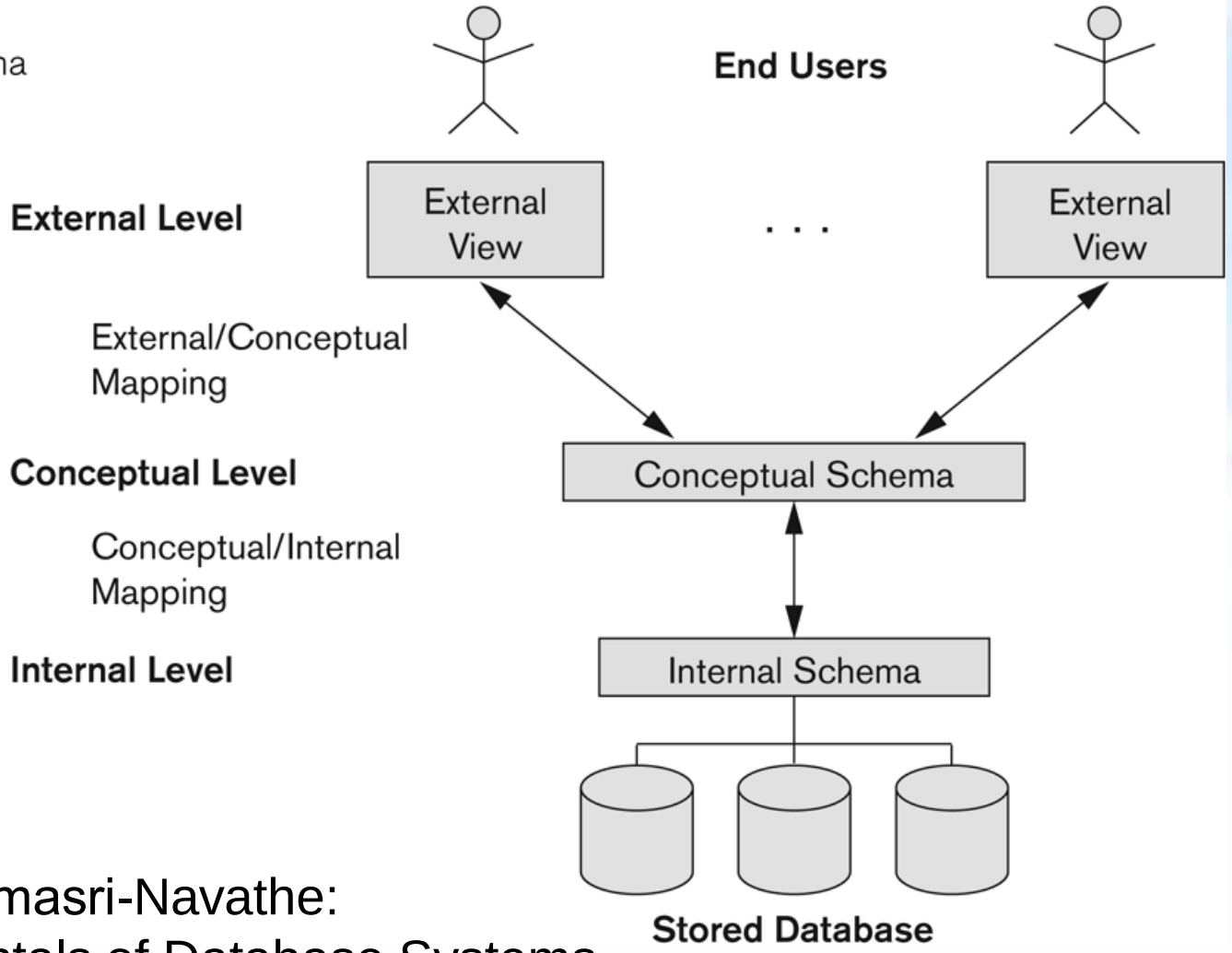
- Az adatmodell a valóság fogalmainak, kapcsolatainak, tevékenységeinek magasabb szintű ábrázolása
  - File-kezelés indexekkel együtt, ezt váltotta fel a
  - CODASYL szabvány, hálós adatmodell (hatékony keresés)
  - Hierarchikus adatmodell (apa-fiú kapcsolatok gráfja)
  - Ted Codd - Relációs adatmodell (táblák rendszere, könnyen megfogalmazható műveletek)
  - Objektum-orientált adatmodell (az adatbázis-kezelés funkcionalitásainak biztosítása érdekében gyakran relációs adatmodellre épül), + Objektum-relációs adatmodell
  - Logikai adatmodell (szakértői rendszerek, tények és következtetési szabályok rendszere)
  - Dokumentumok - Félig strukturált adatmodell, az XML (szabvány adatcsereformaként jelent meg), XML, JSON
  - További lehetőségek, például Gráf adatbázisok, NoSQL

(2) Az adatmodellek 3 szintje

- Hogyan látjuk az adatbázist?
- A 3 szintű ANSI/SPARC architektúra
  - Logikai (külső, a felhasználói szemléletnek megfelelő szinten, nézetek)
  - Fogalmi (conceptual) (absztrakt, szintetizálja az összes felhasználói szemléletet)
  - Fizikai (belső, az adatbázis valamilyen fizikai adatstruktúrában letárolva a háttértárolón)

(2) Az adatmodellek 3 szintje --3

Figure 2.2
The three-schema architecture.



Forrás: Elmasri-Navathe: Fundamentals of Database Systems

(3) Adatbázis-kezelő nyelvek támogatása

- SQL – relációs (és objektum-relációs) adatbázis-kezelő szabvány nyelv, fontosabb szabványok:
  SQL86, SQL89, SQL92 (SQL2), SQL:1999 (SQL3),
  SQL: 2003, SQL:2006, SQL:2008
- DDL (Data Definition Language) adatdefiniáló (sémaleíró) nyelv: sémák, adatstruktúrák megadása, objektumok létrehozása, módosítása, törlése: CREATE, ALTER, DROP
- DML (Data Manipulation Lang.) adatkezelő és lekérdező nyelv: INSERT, DELETE, UPDATE és SELECT
- DCL (Data Control Lang.) adatvezérlő nyelv, jogosultságok kiosztása és visszavonása: GRANT, REVOKE
- Tranzakció-kezelés: COMMIT, ROLLBACK
- Procedurális kiterjesztések: SQL/PSM, Oracle PL/SQL

(4) Több felhasználó támogatása

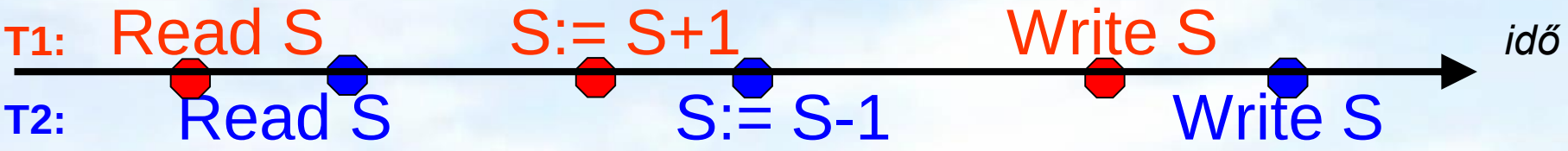
- Felhasználói csoportok. Kulcsemberek:
  - DBA adatbázis-rendszergazda
    - felügyeli az adatbázis-példányokat és adatbázis-szervereket
    - felépíti a rendszert, implementálja és optimális adatbázis-megoldást biztosít
  - Adatbázis-tervező (sématervezés)
  - Alkalmazás-fejlesztő, programozó (kódolás)
  - Felhasználók (akik használják a rendszert)

(5) Adatvédelem, adatbiztonság

- Jogosultságok (objektumok olvasása, írása, módosítása, készítése, törlése, jogok továbbadása, visszavonása) GRANT és REVOKE
- Jogosultságok tárolása rendszertáblákban történik
- Jogosultságok kezelése, felhasználók, jelszavak, hozzáférési jogok
- Adatbázissémák korlátozása (virtuális) nézettáblák segítségével
- Tárolt adatok, hálózati adatforgalmak titkosítása (nagy prímszámok, RSA, DES)

(6) Tranzakció-kezelés

- Tranzakció: adatkezelő műveletekből (adategység írása, olvasása) álló sorozat
- Cél: tranzakciók párhuzamos végrehajtása



- Tranzakció = olyan folyamat, ami adatbázis lekérdezéseket, módosításokat tartalmaz.
- Az utasítások egy „értelmes egészt” alkotnak.
- Egyetlen utasítást tartalmaznak, vagy az SQL-ben explicit módon megadhatóak.

Miért van szükség tranzakciókra?

- Az adatbázis rendszereket általában több felhasználó és folyamat használja egyidőben.
  - Lekérdezések és módosítások egyaránt történhetnek.
  - Az adatbázis rendszereknek el kell különíteniük a folyamatokat.
- Tranzakció = olyan folyamat, ami adatbázis lekérdezéseket, módosításokat tartalmaz.
- Az utasítások egy „értelmes egészt” alkotnak.
- Egyetlen utasítást tartalmaznak, vagy az SQL-ben explicit módon megadhatóak.

A tranzakciók ACID tulajdonságai

- Atomiság (atomicity): a tranzakció egységesen lefut vagy nem, vagy az összes vagy egy utasítás sem hajtódik végre.
- Konzisztencia (consistency): a tranzakció futása után konzisztens legyen az adatbázis, megszorításokkal, triggerekkel biztosítjuk.
- Elkülönítés (isolation): párhuzamos végrehajtás eredménye egymás utáni végrehajtással egyezzen meg
- Tartósság (durability): a befejezett tranzakció eredménye rendszerhiba esetén sem veszhet el

COMMIT és ROLLBACK

- A COMMIT utasítás a tranzakció sikeres befejeződését eredményezi. Egy sikeresen befejeződött tranzakció a kezdete óta végrehajtott utasításainak módosításait tartósan rögzíti az adatbázisban
  - vagyis a módosítások *véglegesítődnek*.
- A ROLLBACK utasítás megszakítja a tranzakció végrehajtását, és annak sikertelen befejeződését eredményezi. Az így befejezett tranzakció SQL utasításai által végrehajtott módosításokat a rendszer meg nem történtekké teszi
  - Vagyis az összes utasítás *visszagörgetésre kerül*, a módosítások nem jelennek meg az adatbázisban.

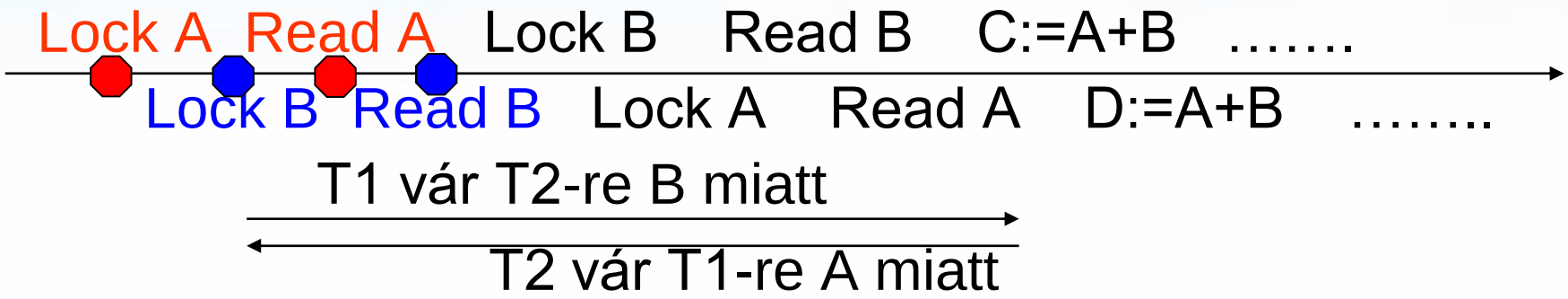
(7) Konkurencia-kezelés

- Zárolások (Lock, Unlock)

T1: (Lock S, Read S, S:=S+1, Write S, Unlock S)

T2: (Lock S, Read S, S:=S-1, Write S, Unlock S)

- A zár kiadásához meg kell várni a zár feloldását.
- Csökken a párhuzamosíthatóság
- Zárak finomsága (zárolt adategység nagysága, zárolás típusa) növeli a párhuzamosíthatóságot
- **Holtpont probléma:**



(8) Naplózás és helyreállítás

- Szoftver- vagy hardverhiba esetén az utolsó konzisztens állapot visszaállítása
- Rendszeres mentések
  - Statikus adatbázis (módosítás nem gyakori)
  - Dinamikus adatbázis (módosítás gyakori)
- Naplóállományok
- Összefügg a tranzakció-kezeléssel

(9) Lekérdezések feldolgozása

1.) Lekérdezésfordító

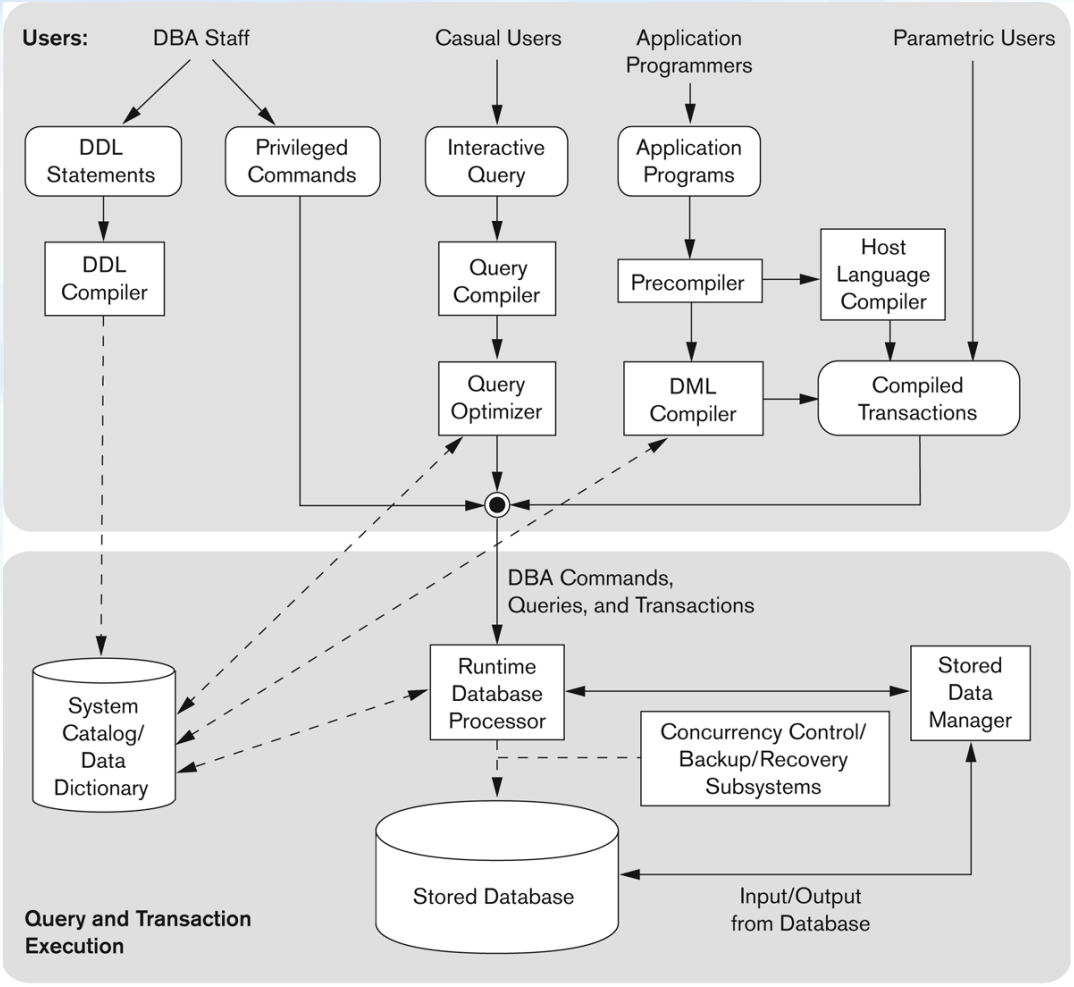
- a.) Lekérdezés-elemző: a lekérdezés szövegéből egy fa struktúrát épít fel. Lekérdezés szintaktikai ellenőrzése.
- b.) Lekérdezés-előfeldolgozó: szemantikai ellenőrzéseket végzi, az adatbázis-objektumok létezésének, és hozzáférési jogoknak az ellenőrzése (metaadatbázis, rendszertáblák). Az elemzőfa átalakítása kezdeti végrehajtási terv legyen.
- c.) Lekérdezés-optimalizáló: Kezdeti végrehajtási tervet átalakítja a lehető legjobb tervvé, vagyis műveletsorozattá a tényleges adatok figyelembevételével. Az adat-struktúrák, méretek statisztikái alapján várhatóan minimális költségű végrehajtási terv kiválasztása.

2.) Végrehajtómotor: Az optimális végrehajtási terv lefuttatása

Összefoglalva: Adatbázis-kezelők részei

- Tranzakció-kezelő:
  - Tranzakciók párhuzamos és biztonságos végrehajtásának a tranzakciók ACID tulajdonságainak biztosítása
- Lekérdezés-feldolgozó
  - Lekérdezés-elemző
  - Lekérdezés-előfeldolgozó
  - Lekérdezés-optimalizáló
  - Végrehajtómotor: az optimális végrehajtási terv lefuttatása
- Tárkezelő és pufferkezelő
  - fizikai adatstruktúrák, táblák, indexek, pufferek kezelése

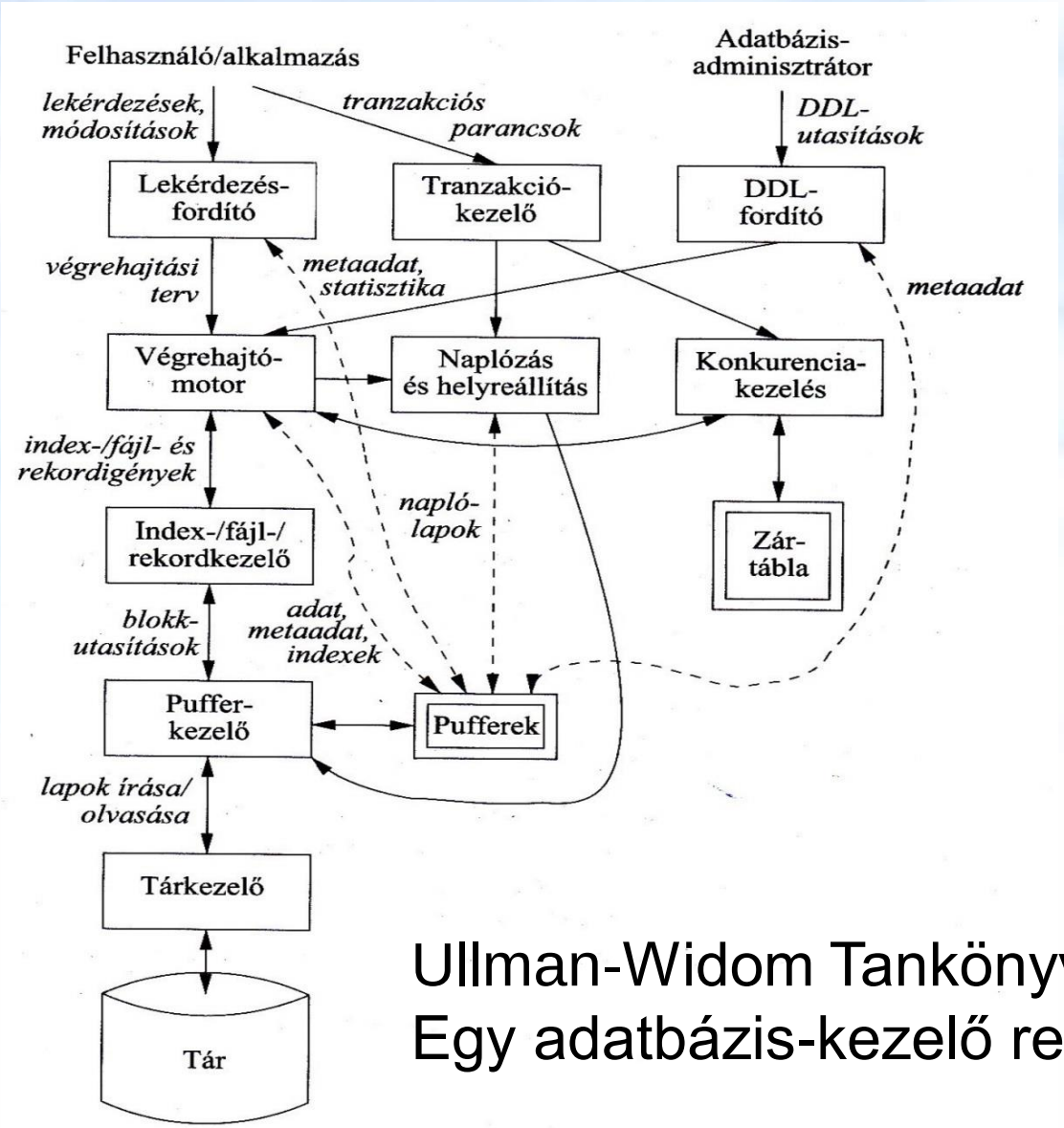
Adatbázis-kezelő rendszer felépítése



Forrás:
Elmasri-Navathe: Fundamentals of Database Systems

Figure 2.3
Component modules of a DBMS and their interactions.

Adatbázis-kezelő rendszer felépítése



Ullman-Widom Tankönyv 1.1. ábra
Egy adatbázis-kezelő rendszer részei

(Tk.7.fej.) Megszorítások (áttekintés)

(1) Kulcsok és idegen kulcsok
- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek
- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások
- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (create assertion)

(6) Triggerek (create trigger)

(Tk.7.4.) Önálló megszorítások Assertions

- SQL aktív elemek közül a leghatékonyabbak nincs hozzárendelve sem sorokhoz, sem azok komponenseihez, hanem táblákhoz kötődnek.
- Ezek is az adatbázissémához tartoznak a relációsémákhoz és nézetekhez hasonlóan.
- CREATE ASSERTION <név>
  CHECK (<feltétel>);
- A feltétel tetszőleges táblára és oszlopra hivatkozhat az adatbázissémából.

Példa: önálló megszorítások

- A Felszolgál(söröző, sör, ár) táblában nem lehet olyan söröző, ahol a sörök átlagára 5 dollárnál több

```
CREATE ASSERTION CsakOlcsó CHECK
 (
NOT EXISTS (
    SELECT söröző
    FROM Felszolgál
    GROUP BY söröző
    HAVING 5.00 < AVG(ár)
));
```

(SELECT .. olyan sörözők, ahol a sörök átlagosan drágábbak 5 dollárnál)

Példa: önálló megszorítások

- Az Sörvivó(név, cím, telefon) és Söröző(név, cím, engedély) táblákban nem lehet több bár, mint amennyi sörivó van.

```
CREATE ASSERTION KevésBár CHECK (
    (SELECT COUNT(*) FROM Söröző)
    <=
    (SELECT COUNT(*) FROM Sörivó)
);
```

Önálló megszorítások ellenőrzése

- Alapvetően az adatbázis bármely módosítása előtt ellenőrizni kell.
- Egy okos rendszer felismeri, hogy mely változtatások, mely megszorításokat érinthetnek.
  - Példa: a Sörök tábla változásai nincsenek hatással az iménti KevésBár megszorításra. Ugyanez igaz a Sörivók táblába történő beszúrásokra is.

(Tk.7.5.) Megszorítások v.s. triggerek

- Aktív elemek – olyan kifejezés vagy utasítás, amit egyszer eltároltunk az adatbázisban és azt várjuk tőle, hogy a megfelelő pillanatban lefusson (pl. adatok helyességének ellenőrzése)
- A megszorítás adatelemek közötti kapcsolat, amelyet az adatbázis-kezelő rendszernek fent kell tartania.
- Triggerek olyankor hajtódnak végre, amikor valamilyen megadott esemény történik, mint például sorok beszúrása egy táblába.

Miért hasznosak a triggerek?

- Az önálló megszorításokkal (assertions) sok mindent le tudunk írni, az ellenőrzésük azonban gondot jelenthet.
- Az attribútumokra és sorokra vonatkozó megszorítások ellenőrzése egyszerűbb (tudjuk mikor történik), ám ezekkel nem tudunk minden kifejezni.
- A triggerek esetén a felhasználó mondja meg, hogy egy megszorítás mikor kerüljön ellenőrzésre.

Triggerek --- 1

- A *triggerek,* amelyeket szokás *esemény-feltétel-tevékenység szabályoknak* is nevezni, az eddigi megszorításoktól három dologban térnek el:
- A triggereket a rendszer csak akkor ellenőrzi, ha bizonyos *események* bekövetkeznek.
  A megengedett események általában egy adott relációra vonatkozó beszúrás, törlés, módosítás, vagy a tranzakció befejeződése.

Triggerek --- 2

- A kiváltó esemény azonnali megakadályozása helyett a trigger először egy *feltételt* vizsgál meg
- Ha a trigger feltétele teljesül, akkor a rendszer végrehajtja a triggerhez tartozó *tevékenységet*. Ez a művelet ezután megakadályozhatja a kiváltó esemény megtörténtét, vagy meg nem történtté teheti azt.

Esemény-Feltétel-Tevékenység szabályok

- A triggereket esetenként *ECA szabályoknak* (*event-condition-action*) *esemény-feltétel-tevékenység szabályoknak* is nevezik.
- Esemény: általában valamilyen módosítás a adatbázisban, INSERT, DELETE, UPDATE.
- Mikor?: BEFORE, AFTER, INSTEAD
- Mit?: OLD ROW, NEW ROW FOR EACH ROW

  OLD/NEW TABLE FOR EACH STATEMENT
- Feltétel : SQL igaz-hamis-ismeretlen feltétel.
- Tevékenység : SQL utasítás, BEGIN..END, SQL/PSM tárolt eljárás

Példa triggerre

- Ahelyett, hogy visszautasítanánk a Felszolgál(söröző, sör, ár) táblába történő beszúrást az ismeretlen sörök esetén, a Sörök(név, gyártó) táblába is beszúrjuk a megfelelő sort a gyártónak NULL értéket adva.

Példa: trigger definíció

```
CREATE TRIGGER SörTrig
  AFTER INSERT ON Felszolgál
  REFERENCING NEW ROW AS ÚjSor
  FOR EACH ROW
  WHEN (ÚjSor.sör NOT IN
     (SELECT név FROM Sörök))
  INSERT INTO Sörök(név)
     VALUES(ÚjSor.sör);
```

Esemény

Feltétel

Tevékenység

Tankönyv példája (7.5. ábra)

```
-- Nem engedi csökkenteni a gyártásirányítók nettó bevételét:
   CREATE TRIGGER NetBevétTrigger
   AFTER UPDATE OF nettóBevétel ON GyártásIrányító
   REFERENCING
        OLD ROW AS RégiSor,
        NEW ROW AS ÚjSor
   FOR EACH ROW
   WHEN(RégiSor.nettóBevétel > ÚjSor.nettóBevétel)
        UPDATE GyártásIrányító
        SET nettóBevétel = RégiSor.nettóBevétel
        WHERE azonosító = ÚjSor.azonosító;
```

Tankönyv példája (7.6. ábra)

-- Az átlagos nettó bevétel megszorítása:

```
CREATE TRIGGER ÁtlagNetBevétTrigger
AFTER UPDATE OF nettóBevétel ON GyártásIrányító
REFERENCING
    OLD TABLE AS RégiAdat,
    NEW TABLE AS ÚjAdat
FOR EACH STATEMENT
WHEN(500000 > (SELECT AVG(nettóBevétel)
                FROM GyártásIrányító)
BEGIN
    DELETE FROM GyártásIrányító
    WHERE (név, cím, azonosító) IN ÚjAdat;
    INSERT INTO gyártásIrányító
        (SELECT * FROM RégiAdat);
END;
```

Tankönyv példája (7.7. ábra)

-- A beszúrt sorok NULL értékeinek helyettesítésére, itt csak egyszerűen 1915-tel helyettesíti a trigger a NULL értéket, de ez akár egy bonyolult módon kiszámított érték is lehet:

(A BEFORE triggerek egy fontos alkalmazása, amikor egy beszúrandó sort a beszúrás előtt megfelelő formára hoznak)

```
CREATE TRIGGER ÉvJavítóTrigger
BEFORE INSERT ON Filmek
REFERENCING
    NEW ROW AS ÚjSor,
    NEW TABLE AS ÚjAdat
FOR EACH ROW
WHEN ÚjSor.év IS NULL
UPDATE ÚjAdat SET év=1915;
```

(Tk. 8.1.) Nézettáblák

- A nézettábla olyan reláció, amit tárolt táblák (vagyis alaptáblák) és más nézettáblák felhasználásával definiálunk.
- **EMPLOYEES table**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	26
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	25
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	105
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	110
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	86
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	83

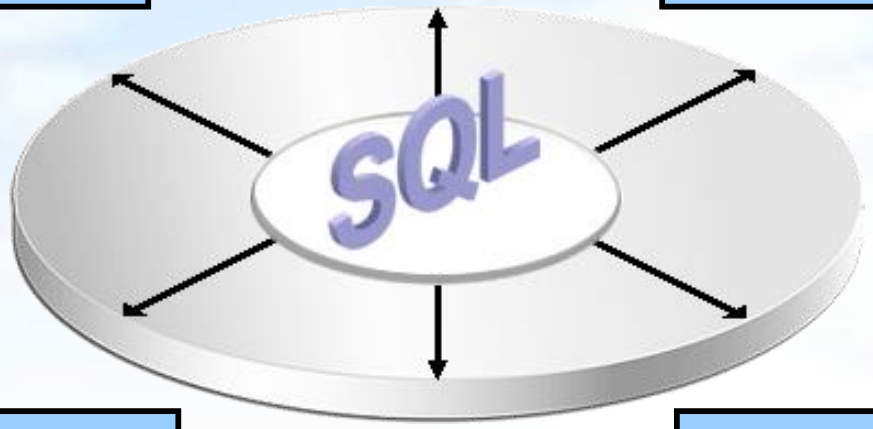
EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

20 rows selected.

A nézettáblák előnyei

Korlátozható az adatok elérése

Bonyolult lekérdezések egyszerűbben kifejezhetők



Az adatfüggetlenség biztosítása

Ugyanazon adatokat különböző nézőpontokból szemlélhetjük

Nézettáblák létrehozása és törlése

- Létrehozása:

```
CREATE [OR REPLACE][FORCE|NOFORCE]
[MATERIALIZED] VIEW <név>
AS <lekérdezés>
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- Alapesetben virtuális nézettábla jön létre.
- Nézettábla megszüntetése:

```
DROP VIEW <név>;
```

Példa: nézettábla létrehozása

- Példa: Egy olyan nézettáblát szeretnénk, mely a Film(cím, év, hossz, színes, stúdióNév, producerAzon) reláció egy részét jelképezi, pontosabban a Paramount stúdió által gyártott filmek címét és gyártási évét

```
CREATE VIEW ParamountFilm AS
    SELECT cím, év
    FROM Filmek
    WHERE stúdióNév = 'Paramount';
```

Példa: nézettáblákhoz való hozzáférés

- A nézettáblák ugyanúgy kérdezhetők le, mint az alaptáblák.
  - A nézettáblákon keresztül az alaptáblák néhány esetben módosíthatóak is, ha a rendszer a módosításokat át tudja vezetni (lásd módosítások, SQL DML)
- Példa lekérdezés:

```
SELECT cím FROM ParamountFilm
WHERE év <= 1990;
```

(Tk.8.2.) Módosítható nézettáblák

- Az SQL szabvány formálisan leírja, hogy mikor lehet egy nézettáblát módosítani és mikor nem, ezek a szabályok meglehetősen bonyolultak.
  - Ha a nézettábla definíciójában a SELECT után nem szerepel DISTINCT, további kikötések:
  - A WHERE záradékban R nem szerepelhez egy alkérdésben sem
  - A FROM záradékban csak R szerepelhet, az is csak egyszer és más reláció nem
  - A SELECT záradék listája olyan attribútumokat kell, hogy tartalmazzon, hogy az alaptáblát fel lehessen tölteni (vagyis kötelező a kulcsként vagy not null-nak deklarált oszlopok megadása)

Nézeteken instead-of-triggerek

Példa: Az előző nézettábla módosításánál, hogy az alaptáblába való beszúráskor a stúdióNév attribútum helyes értéke , 'Paramount' legyen, ezt biztosítja az INSTEAD OF (helyette) típusú trigger:

```
CREATE TRIGGER ParamountBeszúrás
    INSTEAD OF INSERT ON ParamountFilm
    REFERENCING NEW ROW AS ÚjSor
    FOR EACH ROW
    INSERT INTO Filmek(cím, év, stúdióNév)
    VALUES(Újsor.cím, ÚjSor.év, 'Paramount');
```

(Tk.8.5.) Tárolt nézettáblák

- Virtuális vagy materializált? Kétféle nézettábla:
  - Virtuális = nem tárolódik az adatbázisban, csak a relációt megadó lekérdezés.
  - Materializált = kiszámítódik, majd tárolásra kerül.
- **CREATE [OR REPLACE]**
  MATERIALIZED VIEW <név>
  AS <lekérdezés>

Tárolt nézettáblák

- Probléma: minden alkalommal, amikor az alaptáblák valamelyike változik, a materializált nézettábla frissítése is szükségessé válhat.
  - Ez viszont néha túl költséges.
- Megoldás: Periodikus frissítése a materializált nézettábláknak, amelyek egyébként „nem aktuálisak”.

(Tk.9.fej.) Programozási megközelítések

1.) SQL kiterjesztése procedurális eszközökkel, az adatbázis séma részeként tárolt kódrészekkel, tárolt modulokkal (pl. PSM = Persistent Stored Modules, Oracle PL/SQL).

2.) Beágyazott SQL (sajátos előzetes beágyazás EXEC SQL. - Előfordító alakítja át a befogadó gazdanyelvre/host language, pl. C)

3.) Hívásszintű felület: hagyományos nyelvben programozunk, függvénykönyvtárat használunk az adatbázishoz való hozzáféréshez (pl. CLI = call-level interface, JDBC, PHP/DB)

SQL programnyelvi környezetben

- Milyen problémák merülnek fel, amikor egy alkalmazás részeként, programban használjuk az SQL utasításokat?

1.) Osztott változók használata: közös változók a nyelv és az SQL utasítás között (ott használható SQL utasításban, ahol kifejezés használható).

2.) A típuseltérés problémája: Az SQL magját a relációs adatmodell képezi. Reláció: gyűjtemény, sorok multihalmaza, mint adattípus nem fordul elő a magasszintű nyelvekben. A lekérdezés eredménye hogyan használható fel? Megoldás:

Lekérdezések használata a PSM-ben

- A típuseltérés problémája: Az SQL multihalmaz szemlélete hogyan egyeztethető össze a magas-szintű programnyelvekkel? A lekérdezés eredménye hogyan használható fel?
- Három esetet különböztetünk meg attól függően, hogy a SELECT FROM [WHERE stb] lekérdezés eredménye skalárértékkel, egyetlen sorral vagy egy listával (multihalmazzal) tér-e vissza.

Lekérdezések használata a PSM-ben

- SELECT eredményének használata:
  1. SELECT eredménye egy skalárértékkel tér vissza, elemi kifejezésként használhatjuk.
  2. SELECT egyetlen sorral tér vissza
     SELECT e_1, …, e_n INTO vált$_1$, … vált$_n$
     -- A végrehajtásnál visszatérő üzenethez az
     -- SQL STATE változóban férhetünk hozzá.
  3. SELECT eredménye több sorból álló tábla, akkor az eredményt soronként bejárhatóvá tesszük, kurzor használatával.

PL/SQL - Alprogramok

- Tárolt alprogramok
  - Van lehetőség arra, hogy létrehozzunk tárolt eljárást/függvényt
  - Ekkor azt az sqldeveloper eltárolja, később hívható lesz
  - Ez jó az újrafelhasználhatóság szempontjából

PL/SQL - Alprogramok

- Tárolt eljárás létrehozása

```
CREATE [OR REPLACE] PROCEDURE név
[formális paraméterlista]
IS
[deklarációs utasítások]
BEGIN
  végrehajtandó utasítások
  [EXCEPTION kivételkezelő]
END [név];
```

PL/SQL - Alprogramok

- Tárolt függvény létrehozása

```
CREATE [OR REPLACE] FUNCTION név
[formális paraméterlista]
RETURN típus IS
[deklarációs utasítások]
BEGIN
  végrehajtandó utasítások
  [EXCEPTION kivételkezelő]
END [név];
```

PL/SQL - Alprogramok

- Tárolt alprogram újrafordítása

```
ALTER {PROCEDURE | FUNCTION} név
COMPILE [DEBUG];
```

- Tárolt alprogram törlése

```
DROP {PROCEDURE | FUNCTION} név;
```

- Tárolt alprogram meghívása

```
CALL név([aktuális paraméterlista])
[INTO változó];
```

(Tk.10.1) Jogosultság-kezelés

- Egy UNIX-szerű fájlrendszerhez hasonlítva az analógiák: Tipikusan írás, olvasás és végrehajtási jogosultságokról van szó.
- Az adatbázisok lényegesen bonyolultabbak a fájlrendszereknél, ezért az SQL szabványban definiált jogosultágok is összetettebbek.
  - Az SQL kilencféle jogosultságot definiál (SELECT, INSERT, DELETE, UPDATE, REFERENCES, USAGE, TRIGGER, EXECUTE, UNDER)
  - Bizonyos „résztvevőkhöz” sorolja a jogosultságokat, például rendszergazda, korlátozott jogosultságokkal rendelkező felhasználó. Spec. PUBLIC (mindenki)

SQL DCL: GRANT utasítás

- Jogosultságok megadásának szintaktikája:

  GRANT <jogosultságok listája>

  ON <reláció vagy másféle objektum>

  TO <jogosultsági azonosítók listája>;

- Ehhez hozzáadható:

  WITH GRANT OPTION

Példa: GRANT

```
GRANT SELECT, UPDATE(ár)
    ON Felszolgál
    TO Sally;
```

- Ez után Sally kérdéseket adhat meg a Felszolgál táblára vonatkozóan és módosíthatja az ár attribútumot.

Jogosultságok

- A relációkra vonatkozó jogosultságok:

  SELECT = a reláció lekérdezésének joga.

  INSERT = sorok beszúrásának joga.

  (egyetlen attribútumra is vonatkozhat)

  DELETE = sorok törlésének joga.

  UPDATE = sorok módosításának a joga.

  (szintén egy attribútumra is vonatkozhat)

Példa: jogosultságok

- Az alábbi utasítás esetében:

```
INSERT INTO felh.Sörök(név)
SELECT sör FROM felh.Felszolgál f
   WHERE NOT EXISTS
      (SELECT * FROM felh.Sörök
       WHERE név = f.sör);
```

azok a sörök, amelyek még nincsenek benne a sörök táblában. A beszúrás után a gyártó értéke NULL.

- Ehhez az INSERT utasítás végrehajtásához szükséges: SELECT jogosultság a felh (user) felszolgál és sörök tábláira és INSERT jog a Sörök tábla név attribútumára vonatkozóan.

Adatbázis objektumok

- Jogosultságokat nézetekre és materializált nézetekre vonatkozóan is megadhatunk.
- Egy másik fajta jogosultság lehet pl. adatbázis objektumok létrehozásának a joga: pl. táblák, nézetek, triggerek.
- A nézettáblák segítségével tovább finomíthatjuk az adatokhoz való hozzáférést.

Példa: nézettáblák és jogosultságok

- Tegyük fel, hogy nem szeretnénk SELECT jogosultságot adni az Dolgozók(név, cím, fizetés) táblában.
- Viszont a BiztDolg nézettáblán már igen:

```
CREATE VIEW BiztDolg AS
    SELECT név, cím FROM Dolgozók;
```

- A BiztDolg nézettáblára vonatkozó kérdésekhez nem kell SELECT jog a Dolgozók táblán, csak a BiztDog nézettáblán.

Jogosultságok megadása

- A magunk készítette objektumok esetében az összes jogosultsággal rendelkezünk.
- A felhasználókat egy jogosultsági azonosító (authorization ID) alapján azonosítjuk, általában ez a bejelentkezési név, ennek felhasználásával másoknak is megadhatunk jogosultságokat.
- vagy a PUBLIC jogosultsági azonosítót is használhatjuk, a PUBLIC jogosultság minden felhasználó számára biztosítja az adott jogot.
- A WITH GRANT OPTION utasításrész lehetővé teszi, hogy aki megkapta a jogosultságot, tovább is adhassa azt.

Példa: Grant Option

```
GRANT UPDATE ON Felszolgál TO Sally
  WITH GRANT OPTION;
```

- Ez után Sally módosíthatja a Felszolgál táblát és tovább is adhatja ezt a jogosultságot.
  - Az UPDATE jogosultságot korlátozottan is továbbadhatja: `UPDATE(ár)ON Felszolgál`.

Jogosultságok visszavonása

REVOKE <jogosultságok listája>
ON <reláció vagy más objektum>
FROM <jogosultsági azonosítók listája>;

- Az általunk kiadott jogosultságok ez által visszavonódnak.
  - De ha máshonnan is megkapták ugyanazt a jogosultságot, akkor az még megmarad.

(Tk.10.2.) Rekurzió az SQL-ben

Tankönyv

10.2. Rekurzió az SQL-ben: Az Eljut-feladat megoldása

(a.) Datalogban (monoton, lineáris rekurzió)

(b.) SQL-ben WITH RECURSION utasítással

Oracle kiegészítések - az Eljut feladat megoldása:

(c.) Eljut feladat PL/SQL-ben (illetve SQL/PSM-ben)

(d.) Oracle megoldások/új: WITH alkérdés faktorizáció

(e.) Oracle megoldások/régi: CONNECT BY PRIOR

Az Eljut-feladat

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- Jaratok(legitarsasag, honnan, hova, koltseg, indulas, erkezes) táblában repülőjáratok adatait tároljuk.
- A járatok táblát létrehozó script:

http://sila.hajas.elte.hu/AB1gy/create_jaratok_tabla.txt

- Mely (x,y) párokra lehet eljutni x városból y városba?
- Ezt egy relációs algebrai kifejezésként nem tudjuk megadni zárt alakban, klasszikus SQL SELECT utasítással sem tudjuk kifejezni, csak azt tudjuk, hogy átszállás nélkül, egy, két, stb… átszállással:

Az Eljut-feladatnak nincs algebrai megoldása

```
select distinct honnan, hova
    from jaratok
union
select j1.honnan, j2.hova
    from jaratok j1, jaratok j2
    where j1.hova=j2.honnan
union
select j1.honnan, j3.hova
    from jaratok j1, jaratok j2, jaratok j3
    where j1.hova=j2.honnan
    and j2.hova=j3.honnan
--- union stb... Ezt így nem lehet felírni…
```

Az Eljut-feladat Datalogban

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- Jaratok(legitarsasag, honnan, hova, koltseg, indulas, erkezes) EDB-táblában repülőjáratok adatait tároljuk.

  Mely (x,y) párokra lehet eljutni x városból y városba?

- Datalogban felírva (lineáris rekurzió)

  Eljut(x, y) ← Jaratok(l, x, y, k, i, e)
  Eljut(x, y) ← Eljut(x, z) AND Jaratok(l, z, y, k, i, e)

- Vagy másképp felírva Datalogban (mi a különbség?)

  Eljut(x, y) ← Jaratok(_, x, y, _, _, _) -- anonimus változók
  Eljut(x, y) ← Eljut(x, z) AND Eljut(z, y) -- nem lineáris rek.

Az Eljut feladat SQL-99 szabványban

- Datalog LINEÁRIS, MONOTON rekurzió átírható:
  Eljut(x, y) ← Jaratok(l, x, y, k, i, e)
  Eljut(x, y) ← Eljut(x, z) AND Jaratok(l, z, y, k, i, e)

- Hova, mely városokba tudunk eljutni Budapestről?

```
WITH RECURSIVE Eljut(honnan, hova) AS
        (SELECT honnan, hova FROM Jaratok
      UNION
          SELECT Eljut.honnan, Jaratok.hova
          FROM Eljut, Jaratok
          WHERE Eljut.hova = Jaratok.honnan)
SELECT hova FROM Eljut WHERE honnan='Bp';
```

SQL-99 szabvány: Rekurzív lekérdezés

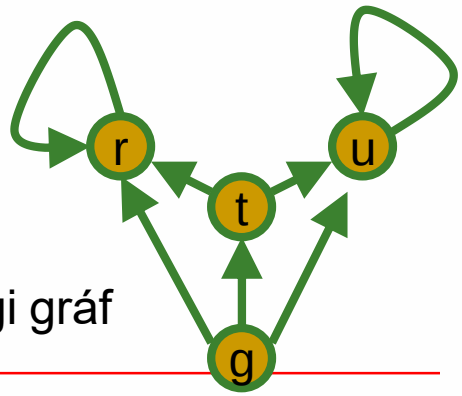
- A WITH utasítás több ideiglenes relációra vonatkozó definíciója:

WITH [RECURSIVE] R_1 AS <R_1 definíciója>
[RECURSIVE] R_2 AS <R_2 definíciója>
...
[RECURSIVE] R_n AS <R_n definíciója>
< R_1,R_2,...,R_n relációkat tartalmazó lekérdezés>

Másik példa: Rekurzív Datalog

- A testvérek (féltestvérek) gyerekei első unokatestvérek, az első unokatestvérek gyerekei másod-unokatestvérek, és így tovább. Hívjuk egyszerűen unokatestvéreknek, akik valamilyen szinten unokatestvérek. A rokonok azok, akik közös ősnek leszármazottjai.
- Milyen Datalog program írja ezt le?

testvér(x,y) ←gyerek(x,z),gyerek(y,z),x ≠y
unokatestvér(x,y) ←gyerek(x,z),gyerek(y,v),testvér(z,v)
unokatestvér(x,y) ←gyerek(x,z),gyerek(y,v),unokatestvér(z,v)
rokon(x,y) ←testvér(x,y)
rokon(x,y) ←rokon(x,z),gyerek(y,z)
rokon(x,y) ←rokon(z,y),gyerek(x,z)



Függőségi gráf

Példa folyt:Rekurzív Datalog átírása SQL-be

```
WITH
  T(x,y) as (select G1.u x, G2.w y from G G1,G G2
        where G1.w=G2.u and G1.u<>G2.u),
  RECURSIVE U(x,y) as (select G1.u x, G2.u y
        from G G1, G G2, T where T.x=G1.w and T.y=G2.u)
                UNION (select G1.u x, G2.u y
        from G G1, G G2, U where U.x=G1.w and U.y=G2.u),
  RECURSIVE R(x,y) as (select * from T) UNION
      (select R.x x,G.u y from R,G where R.y=G.w) UNION
      (select G.u x, R.y y from R,G where R.x=G.w)

 (select T.x, T.y, 'T' from T union
  select U.x, U.y, 'U' from U union
  select R.x, R.y, 'R' from R);
```

Rekurzív lekérdezések

- Datalog rekurzió segít megérteni az SQL-99 szabványban bevezetett rekurzív lekérdezések WITH RECURSIVE záradékát.
- A BSc-n csak MONOTON rekurziót vesszük, vagyis nem használjuk nem-monoton különbség műveletet, nincs csoportosítás-aggregálás (ugyanis az olyan lekérdezések, amelyek nem-monotonok, megengedik a negációt és aggregálást az olyan különös hatással van a rekurzióra, ezt csak MSc kurzusokon vesszük).
- A rekurzív Eljut-feladatnak az Oracle CONNECT BY záradékkal ill. az egyéb megoldásait is megnézzük!

Eljut feladat PL/SQL-ben ---1

- **Rek1.feladat:** Mely (x, y) várospárokra lehet egy vagy több átszállással eljutni x városból y városba?
- Ehhez hozzuk létre eljut(honnan,hova) táblát,

```
DROP TABLE eljut;
CREATE TABLE eljut(
        honnan VARCHAR2(10),
        hova VARCHAR2(10));
```

- Írjunk egy olyan PL/SQL programot, ami feltölti az ELJUT táblát a sorait a járatok tábla alapján (ehhez ciklust szervezni, az insert több sor felvitele 2.alakja alkérdéssel járatok és eljut táblák alapján)

Eljut feladat PL/SQL-ben ---2

- Az ELJUT feladat megoldása Oracle PL/SQL-ben
- A ciklus során ellenőrizni kell, hogy addig hajtsuk végre a ciklust, amíg növekszik az eredmény (Számláló)
- DECLARE RegiSzamlalo Integer;
  UjSzamlalo Integer;
- Deklarációs rész után BEGIN … END; között az utasítások, először az eljut táblának kezdeti értéket adunk (a megvalósításnál az INSERT-nél figyelni, hogy ne legyenek ismétlődő sorok: select distinct)

  delete from eljut;

  insert into eljut (SELECT distinct honnan, hova
  FROM jaratok);

Eljut feladat PL/SQL-ben ---3

- Szamlalo változóknak adunk kiindulási értéket:

```
RegiSzamlalo := 0;
select count(*) into UjSzamlalo from eljut;
```

- A ciklust addig kell végrehajtani, amíg növekszik az eredmény (Szamlalo) duplikátumokra figyelni!

```
LOOP
   insert into eljut (lásd a köv.oldalon…)
   select count(*) into UjSzamlalo from eljut;
   EXIT WHEN UjSzamlalo = RegiSzamlalo;
   RegiSzamlalo := UjSzamlalo;
END LOOP;
commit;
```

Eljut feladat PL/SQL-ben ---4

- Az eljut tábla növelése a ciklusban, figyelni kell a duplikátumokra, csak olyan várospárokat vegyünk az eredményhez, ami még nem volt!

```
insert into eljut
    (select distinct eljut.honnan, jaratok.hova
     from eljut, jaratok   --- *from (lineáris rekurzió)
     where eljut.hova = jaratok.honnan
     and (eljut.honnan,jaratok.hova)
          NOT IN (select * from eljut));
```

- Megjegyzés: PSM-ben a nem-lineáris rekurzió is megengedett: from eljut e1, eljut e2 ---*from-ban

Eljut feladat PL/SQL-ben ---5

- **Rek2.feladat:** Mely (x,y) város párokra hány darab átszállással és milyen költségekkel lehetséges egy vagy több átszállással eljutni x városból y városba?
- Ehhez készítsünk Eljut2(honnan, hova, atszallas, koltseg) táblát. Írjunk egy olyan PL/SQL programot, ami feltölti az ELJUT táblát.

- **Rek3.feladat:** Tegyük fel, hogy nemcsak az érdekel, hogy el tudunk-e jutni az egyik városból a másikba, hanem az is, hogy utazásunk során az átszállások is ésszerűek legyenek, ami azt jelenti, hogy ha több járattal utazunk, akkor nézni kell átszálláskor az érkező járatnak legalább egy órával a rákövetkező indulás előtt meg kell érkeznie, és 6 óránál ne kelljen többet várnia.

Oracle megoldások: with utasítással

- Az Oracle SQL a WITH RECURSIVE utasítást (UNION) nem támogatja, ott másképpen oldották meg WITH utasítással (Oracle 11gR2 verziótól használható)

```
WITH eljut (honnan, hova) as
   (select honnan, hova from jaratok
  UNION ALL
     select jaratok.honnan, eljut.hova
     from jaratok, eljut
     where jaratok.hova=eljut.honnan
     )
SEARCH DEPTH FIRST BY honnan SET SORTING
CYCLE honnan SET is_cycle TO 1 DEFAULT 0
select distinct honnan, hova from eljut order by honnan;
```

Oracle megoldások: connect by

- SELECT DISTINCT hova FROM jaratok
  WHERE HOVA <> 'DAL'
  START WITH honnan = 'DAL'
  CONNECT BY NOCYCLE PRIOR hova = honnan;

- SELECT LPAD(' ', 4*level) || honnan, hova,
  level-1 Atszallasok,
  sys_connect_by_path(honnan||'->'||hova, '/'),
  connect_by_isleaf, connect_by_iscycle
  FROM jaratok
  START WITH honnan = 'SF'
  CONNECT BY NOCYCLE PRIOR hova = honnan;

Kérdés/Válasz

- Köszönöm a figyelmet! Kérdés/Válasz?
- Próbáljuk ki az Oracle gépes-megoldásait az Eljut-feladathoz, ehhez a Jaratok táblát létrehozó script:

  http://sila.hajas.elte.hu/AB1gy/create_jaratok_tabla.txt
- Vizsgára: SQL és PL/SQL gyakorlás sqldeveloperben

  Kende Mária - Nagy István: Oracle Példatár feladatai