Logó, ha szükséges

ADATBÁZISRENDSZEREK

Dr. Radványi Tibor

2013.02.13

Tartalomjegyzék

Bevezetés

Adat és információ. A XX, XXI. század, illetve az elmúlt 50 év legújabb és ma már legfontosabbnak tekinthető fogalmai közé tartoznak, mivel szinte mindennapi életünk részévé váltak. Egyre több és több, újabb és újabb információ ér bennünket; társadalmunkban felértékelődött az információ szerepe és értéke. Az információ a külvilágból folyamatosan ér bennünket, szinte bombáznak vele: a televízióból, a rádióból, az újságból híreket hallunk, embertársainktól értesülést szerzünk a legfontosabb eseményekről szinte a nap majdnem 24 órájában folyamatosan. A hatalmas információtömegből igyekszünk a legfontosabbakat, a számunkra legfontosabbakat kiemelni és ezekre összpontosítani. Ami azért fontos, mert ilyen mennyiségű információt lehetetlen észben tartani. Néha a keveset sem tudjuk, vagy épp nem szeretjük megjegyezni. Ezért az információt valahogyan máshogyan kell rögzítenünk, nem a fejben. A szó elszáll, az írás megmarad, ahogy a szállóige szól. Emellett mindenkinek érdeke, hogy a rögzített információhoz gyorsan hozzá tudjon jutni.

Az információ egyenlő hatalom, ahogy a mondás tartja. És ez bizony igaz is. Nem kell hosszan ecsetelnem, mennyire fontos, például bankkártyánk adatait megfelelő helyen tartani. Fontos ezért, hogy információinkat biztonságosan tudjuk tárolni, erre megtaláljuk a megfelelő megoldást. Ki kell tehát találni olyan eljárást, mellyel mindez könnyen, egyszerűen és gyorsan megvalósítható.

Könnyen belátható, hogy a fenti ismeretekkel főként az általános, a további ismeretekkel pedig a középiskolai tanulók megismerkedjenek tanulmányaik (és természetesen életük) során. A Nemzeti Alaptanterv fejlesztési követelményei között az *alkalmazói ismeretek* szerepe és fontossága kiemelkedő óraszámot kap, ezért tanítására pedagógusainknak különös figyelmet kell fektetniük. Külön szerepet kap ebben a keretben az adatbázisokkal, adatbázis kezelő rendszerekkel való megismerkedés fontossága a fent említett okok miatt. Emellett, a tanulóknak (és persze mindenkinek) fontos, hogy lépést tudjanak tartani azokkal a változásokkal és célokkal, melyek az informatikát fejlődésében érintik, s a megújulásokkal lépést tudjanak tartani. Emellett az informatikával foglalkozó személyek száma és képzettségének szintje nagymértékben emelkedik. Tartsunk lépést mindkettővel, hiszen egyre nagyobb a kereset és az igény a magasan képzett szakemberekre.

A számítástechnika egyik fontos jellemzője, hogy egyre több felhasználó egyre több számítógépen tárolt adatot használ fel. Az elkészített és alkalmazott számítógépes

programrendszereknek növekvő adatmennyiséggel kell megbirkózniuk. Hétköznapjainkban mind gyakrabban találkozhatunk a számítógépes információs rendszerek alkalmazásával.
Számítógépes információs rendszereket használnak az üzemekben a termelés irányítására, pénzügyi, személyzeti, raktári, anyaggazdálkodási feladatok elvégzésére. Néhány alkalmazási terület az élet minden területéről említhető:

- Kereskedelem: raktárkészlet nyilvántartás
- Közigazgatás: adónyilvántartás
- Egészségügy: betegnyilvántartás
- Közlekedés: helyfoglalási rendszerek, menetrendek
- Mérnöki munka: tervezői rendszerek
- Oktatás: tanulói nyilvántartás

Mindezekben az a közös jellemző, hogy nagy adathalmazt kezelnek, az adatok közt bonyolult kapcsolatok is fennállhatnak, és ezeket az adatokat hosszabb ideig is meg kell őrizni.
Ezeken kívül más fontos sajátosságokkal is rendelkeznek ezek a rendszerek, de vannak követelmények, melyeknek feltétlenül teljesülniük kell:

- Nagymennyiségű adat hatékony kezelése
- Egyszerre több felhasználó általi elérés támogatása
- Integritásőrzés
- Védelem

Hatékony programfejlesztés

Alapfogalmak

Az első adatbázis-kezelő rendszerek a fájlkezelő rendszerekből kezdtek kialakulni a 60-as évek végén. Ezek terjedelmes és drága programrendszerek voltak, melyek nagyméretű gépeken futottak. Azok a rendszerek voltak az első jelentőségteljes alkalmazási területei, melyekben sok adatelemet tároltak és a rendszerben sok lekérdezést és módosítást hajtottak végre. Néhány példa: Vállalati nyilvántartások, banki rendszerek, repülőgép-helyfoglalási rendszerek.

1970-ben Ted Codd cikkének közzététele után – amiben azt javasolta, hogy az adatbáziskezelő-rendszerek az adatokat táblázatok formájában jelenítsék meg a felhasználó felé – jelentősen megváltoztak az adatbáziskezelő rendszerek.

A különbség a korábbi rendszerekkel szemben, hogy a felhasználónak nem kell az adatok tárolási struktúrájával foglalkoznia a relációs rendszerben, ugyan is a lekérdezések egy olyan magas szintű nyelv közvetítésével fejezhetők ki, melyek alkalmazása igen csak megnöveli az adatbázis programozók hatékonyságát.

Ebben a modellben nincsenek kitüntetett adatok, azaz a halmaz elemeiről nyilvántartott tulajdonságok egyenrangúak. A rendszer ez által sokkal rugalmasabban használható, mivel a keresési feltételek szinte tetszőlegesen megfogalmazhatók.

Az IBM volt az első olyan cég, mely relációs modell előtti, és a relációs modelleket támogató adatbáziskezelő-rendszert is árusított.

Napjainkra a relációs modellen alapuló adatbázisrendszerek ugyanolyan elterjedt számítógépes eszköznek számítanak, mint korábban a szövegszerkesztő, táblázatkezelő programok.

Az adat és információ

Az információ mai világunk főszereplője. Ha korunkat, társadalmunkat jelzős szerkezettel akarjuk ellátni, magától adódik az információs társadalom kifejezés. Fontos kérdés, hogy tudjuk-e valójában mi is az információ? Teljes mértékben elfogadható definíciót még nem találtak rá, bár minden vele foglalkozó szakterület megalkotta a saját nézőpontjából fontos tulajdonságokkal rendelkező entitásukat, melyet információ névvel láttak el.

Az **információ** észlelt, érzékelt, felfogott és a fogadó számára szükséges, újdonságot jelentő adat, amit megszerzett ismereteinktől függően értelmezünk.
Az **adat** fogalmán valamilyen tény megjelenését értjük, amit rögzíteni, tárolni, átalakítani és továbbítani tudunk. Az adat fogalma igencsak tág. Az adatbázis-kezelés szempontjából az adat a számítógépen tárolt információ nélküli jelsorozat, amelyből a feldolgozás során nyerhetünk információt.
Adatbázis-kezelés szempontjából az adat a számítógépen tárolt jelsorozat, amelyből a feldolgozás során nyerhetünk információt. Addig viszont információ nélküli jelsorozat.
Ha adatokat gyűjtünk össze, és azokat a közöttük fennálló összefüggésekkel együtt egy helyen tároljuk, akkor az így létrehozott együttest adatbázisnak nevezzük:
Például: betegek kartonjai a kórházban, autók adatai a rendőrségen, egy telefonkönyv bejegyzései.

Az adatbázis

Adatbázis az integrált, logikailag összetartozó információk összessége; az adatok és a köztük lévő összefüggések rendszere, melyet egymás mellett tárolunk. Ahhoz, hogy a későbbiekben hatékonyan tudjunk dolgozni az adatbázissal, fontos, hogy jól megtervezzük a szerkezetét.
Az **adatbázis rendszer** magába foglalja az adatbázisokat, a számítógépes erőforrásokat, sőt tágabb értelemben ide vehetjük az adatbázis-adminisztrátorokat, akik azok a személyek, akik az adatbázisok kezelését, tervezését végzik.

Az adatbázis kezelő rendszerek (ABKR/ DBMS)

Az adatbázis egy olyan adatgyűjtemény, amely szervezett módon tárolja az adott feladathoz kapcsolódó adatokat, gondoskodik az adatokhoz való hozzáférésről, mindemellett az adatok integritásának megőrzését, és az adatok védelmét biztosítja.
Az adatbázis-kezelő rendszerek megkönnyítik az adatbázisok kezelését. Feladatuk az adatbázisban lévő adatok rögzítése, tárolása, kezelése.
Az ANSI/SPARC modell a felhasználó és a számítógépes háttértárolón fizikailag tárolt adatok közötti kapcsolatot szemlélteti. Ez alapján három szintet különböztetünk meg:

- Külső szint, más néven felhasználói nézet, mely a felhasználó szemszögéből vizsgálja az adatokat.
- Koncepcionális szint, mely magába foglalja az összes felhasználói nézetet. Az adatbázist ezen a szinten a logikai sémával adják meg.

- Belső szint, más néven fizikai szint, ez az adatoknak az adott számítógépes rendszerben való aktuális bemutatását jelenti.

Amikor az ANSI/SPARC modellről beszélünk, fontos megemlítenünk két fogalmat. A logikai és a fizikai adatfüggetlenséget.

A fizikai adatfüggetlenség azt jelenti, ha a belső szinten változtatást hajtunk végre, az nincs hatással a logikai sémára, így azon nem kell változtatásokat végrehajtanunk. Tehát, ha az adatok tárolásában változás történik, az nincs hatással a felette lévő szintekre. A külső és a koncepcionális szint közötti adatfüggetlenség a logikai adatfüggetlenség.

Adatbázis-kezelő rendszernek nevezik az olyan programrendszereket, melynek feladata az adatbázishoz történő hozzáférések biztosítása és az adatbázis belső karbantartási feladatainak ellátása, azaz:

- Adatbázisok létrehozása
- Adatbázisok tartalmának definiálása
- Adatok tárolása
- Adatok lekérdezése
- Adatok védelme
- Adatok titkosítása
- Hozzáférési jogok kezelése
- Fizikai adatszerkezet szervezése

Amit szem előtt kell tartanunk, az, hogy az adatbázis rendszerek architektúrája hogyan változott, hogyan tudjuk összerakni ezeket. Ezek fontosak mindenkinek. Főleg a PTI-seknek különösen, mert ők abban a helyzetben vannak, hogy kapnak egy megrendelést és dönteniük kell tegyük fel arról, hogy mit használnak. Mert nem az a jó programozó, programfejlesztő, aki tud egy féle adatbázis kezelő programot használni, tud egy nyelven programozni. Hát ez az általános iskola. Az a jó, hogyha kapsz egy feladatot és el tudod dönteni, hogy melyik irányvonalon kell elindulni. Milyen adatbázis kezelőt kellene használni, és milyen nyelven programozol. Persze nem lehet azt mondani, hogy azt a néhány 100 programozási nyelvet mindenki tudja fejből, mert ez nyilván nem igaz, kettőt – hármat fogunk elővenni, de nagyon jól tudja mindenki, aki már próbált weblapot készíteni, hogy nem biztos, hogy érdemes egy weblapfejlesztésnek nekiugrani, mondjuk egy aspx.net-el. Lehet, hogy egy nagyobb feladatnál igen, de lehet, hogy egy kisebbet az ember összerak egy html kóddal és nem is visz bele semmiféle dinamizmust, vagy esetleg php-ban egyszerűbben megoldhatók a dolgok. Ezek

specifikus dolgok. Visszatérve az adatbázis architektúrákra, a kérdés az, hogy az egyes adatbázis–kezelők milyen környezetben tudnak teljesíteni még nekünk, mert nem igaz az, hogy bármely adatbázis–kezelő bármely környezetben ki tudja szolgálni a mi igényeinket.

Lokális adatbázisok

Az első ilyen architekturális szint a lokális adatbázisok: ezek a „legjobbak", egy gép, egy adatbázis, egy felhasználó, senkinek nincs semmi gondja. Innen indult a mese, valamikor 1980 körül, amikor kezdtek a személyi számítógépeken megjelenni az adatbázis–kezelők. dBase világ, DOS alapokon (a DOS eleve nem adott lehetőséget több felhasználóra, több szálon futtatásra). Itt nem voltak olyan problémák, hogy hálózati ütközés, konkurens hozzáférés. Ilyen adatbázisok voltak a dBase 3, 4, 5; ennek a továbbfejlesztett változata a Paradox 4, 5, 7; aminek stabilabb volt az adattábla–kezelése, de cserébe kaptunk egy sérülékeny indextáblát. Mindegyikre jellemző volt, hogy egy adattábla – egy fájl; egy index – egy fájl; egy leíró tábla – egy fájl, check-feltételek egy táblához – egy fájl. Ha volt egy adatbázisom 100 táblával, akkor egy könyvtárban néhány 100 darab fájl jött létre és ezeket kezelte egy adatbázis–kezelő motor. Fájl szinten dolgozott, bájtokat mozgatott és blokkokat kezelt. Mivel fájl szinten nyúlt hozzá, sérülékeny volt. Sok fájl volt, ezért sok sérülési lehetőség volt, sok törlési lehetőség volt már az operációs rendszer szintjén is. Ha jött egy áramszünet, hívni kellett a programozót, mert fejre állt az egész rendszer. Valamit valamiért. Azért mindig azt szoktam mondani, hogy ezek a rendszerek veszélyes rendszerek, főleg hogyha nem lokális adatbázis rendszerben használjuk őket. Manapság már nagyon nehéz lenne lokális adatbázist használni. Ide szokták az MSAccess-t is sorolni. Ez annyival modernebb, hogy az összes eszközt, adatot és leíró eszközt egy fájlban tárolja. Innen kezdve ugyanazt tudja, mint a Paradox vagy a dBase. Nagyon sérülékeny tud lenni, ha nagyobb terheléssel akarjuk használni. Tanításra kiváló eszköz (ECDL, érettségi). A LibreOffice-nak is ott van a Base adatbázisa, az hasonló az Accessshez, ingyenes, ismerkedésre és tanulásra való. Ezeknek az adatbázis kezelőknek vannak korlátaik: egy forgalmi táblában folyamatosan növekszik a rekordok száma. Könnyedén elérheti a 100 000 menniységet. Lehet, hogy soknak tűnik, de ha valaki ír egy rendszert, amit használnak is, kiderül hogy inkább nagyon kevés. Nem lehet azt mondani, hogy 100 ezerig jól működik, 100 001-nél beomlik az egész. 2 – 300 ezer rekordig működik, aztán egyre többször jönnek elő hibák, elkezd lassulni a rendszer, indexsérülések jönnek elő, tehát a lokális, fájl-alapú rendszereknek a teljesítőképessége kb. 100 000-es nagyságrendű. Ha tudjuk ezt, és ha tudjuk, hogy milyen feladattal akarnak

megbízni, akkor nincsen gond, hogy ezt használjuk. Ha Mariska néninek kell a virágboltjába csinálni egy adatbázist, amibe hetente beviszi, hogy kapott 10 szál tulipánt és 30 szál rózsát, meg eladott 9 szál tulipánt meg 34 szál rózsát és semmi mást nem visz be, arra kiváló az Access. Ne próbáljuk rábeszélni a Mariska nénire az Oracle-t.

File – server architektúra

Természetesen fejlődött a világ, van kábel, összekötjük a gépeket, akármennyit, de hát az volt a gond ezzel, hogy az adatbázis-kezelés gyerek volt még. Úgyhogy kifejlesztették ezt a csodálatos file – server architektúrát. Itt zárójelben megjegyzem, bár nem kizárólagos a Novell szerver használata, de akkor volt a fénykora. Nem olyan rég volt ez, kb. 15 éve. Viszont informatikában sok idő. Nagyon jó, jól kidolgozott, robosztus rendszerek voltak, de „fájl-szerver", már a nevében benne van, hogy arra való, hogy a dokumentumokat, fájlokat osszon meg, mint erőforrásokat. A Novell-t ráerőszakolták az adatbázis-kezelésre. Fogták, ezeket berakom a Novell szerver alá, vagy a Windows szerver alá megosztott mappába, és akkor majd az operációs rendszer biztosítja azt, hogy ki érheti el, ki nem érheti el. Aztán persze nem működött, mert nem volt joga írni, akkor meg kellett a jogot adni hozzá, kiderült, hogy igazándiból akkor működik, hogyha admin jogot adunk abba a könyvtárba. Elkezdtük a lokális adatbázis fájlokat megosztani a hálózaton. Innen kezdődtek a problémák. A userek egyszerre akarták ugyanannak a táblának ugyanazt a rekordját módosítani. Előjött a konkurens hozzáférés problémája. Meg kellett oldani, elkezdtük patkolni az adatbázis kezelőket, kitaláltunk a dBase-hez egy programozói felületet, Meg tudták mondani, hogy zárolják az adattáblát, vagy az egész adatbázist. Enyém, senki másé. Dolgozok benne, ha végeztem, felszabadítom, és Te is hozzányúlhatsz. Ja, elfelejtettem, majd holnapután felszabadítom. Sok probléma forrása volt a nem megfelelően finom szemcsézettségű zárolás. Ráadásul ez nem a rendszer része volt, hanem a programozó vezérelte. Ennek ellenére sok-sok évig használták. Karakteres felületen programozva, viszonylag gyors volt ez az adott körülmények között, de hát persze a megbízhatóságot sok kívánnivalót hagyott: a konzisztens adatbázis fogalom, ami még megvolt lokálisan, azt el is felejthetjük (tehát konzisztens volt az adatbázis üresen). Egyszerűen nem volt eszköz a konkurens hozzáférés kezelésére. Voltak próbálkozások, hogy amikor az egyik felhasználó dolgozik vele, akkor lemásolja, a teljes adatbázist magának, abban dolgozott, kinaplózta a különbségeket, amiket ő csinált, és amikor visszatöltötte, csak különbségeket vitte a napló alapján föl. De ezt csak este lehetett, amikor más nem nyúlt az adatbázishoz.

Kliens – szerver architektúra

Két oldala van, az egyik a hardveres architektúra, amikor azt mondom, van egy szerverem, és vannak a kliens gépek. A szerver valamilyen szolgáltatást nyújt, a kliens meg ezt használja. Csakhogy mi adatbázisról beszélünk. Általában mindenki egy nagy böhöm számítógépre gondol, ami a szerver és néhány kis laptopra, amik a kliensek. De az adatbázis–kezelésnél ez nem igaz. Itt az a szerver, amelyik az adatbázis szolgáltatást nyújtja és az a kliens, aki az adatbázis szolgáltatást igénybe veszi. Ha Pistike számítógépén van egy MS SQL szerver én meg huncut módon bejelentkezek hozzá és használom akár a hozzá rendelt adatbázisok elérésére, az ő laptopja a szerver, az enyém meg a kliens. Eszembe jut, hogy valamit nem jól csinált meg. Nézd már meg az enyémet! Ő bejelentkezik az én SQL szerveremre, az én adatbázisomba, akkor az én laptopom lesz a szerver és az övé a kliens. Tehát az, hogy ki a szerver és ki a kliens, a szolgáltatáson múlik. Ezek tanpéldák, de amikor egy nagy cégnél dolgozunk, ott azért a hardverhez igazodnak a szolgáltatások is. Egyszerűen azért, mert nagyobb erőforrásra van szükség ahhoz, hogy néhány száz, néhány ezer ember kérését kiszolgálja egy kiszolgáló. Ezért a szerver számítógépen fut egy SQL szerver, mint szolgáltatás, itt pedig kliens programok. Még mindig absztrakt szinten vagyunk: mi az, hogy SQL szolgáltatás? Odamegy valaki a MS Windows 2008 szerverhez és azt mondja, hogy kérek tőled egy szolgáltatást. Azt mondja erre a Windows, hogy mi van!? Ilyenem nincs. Odamegy a Unix-hoz, az is azt mondja neki, hogy mi van!? Ilyenem nincs. Nincsen az operációs rendszerekben ilyen, ez egy külön mese, milyen szoftvereket, milyen szervereket, milyen szolgáltatásokat telepítünk. Ami ezt tudja, az két nagy csoportra szedném szét. Érdekes dolog, hogy Magyarországon az Oracle a legelismertebb „adatbázis–kezelő”, Romániában azt mondják, hogy igen, van Oracle, de az IBMDB2 az „adatbázis–kezelő”. Marketing, semmi más. Mivel fizetős – és nem is kicsit – azért ide írom az MS SQL-t. Nagyon szép SQL szerver, én azt szoktam mondani, hogy a Microsoft legjobb terméke. Tud robosztus lenni, tud jól működni. Azért ide sorolom az IBM DB2-t és a Sybase-t is. A fizetősök közé kell még sorolnom az InterBase-t. Egy verziója volt, a hatos, ami ingyenes volt. A Borland cég kiemelt partnere, a Delphi és a C++ Builder kiváló SQL szervere lehet.
Ezek az SQL szerverek jó pénzért megvásárolhatók. EZ a díj pár 100 000-től több 10 millióig is felmehet. Tehát amikor Mariska néni kisboltjába írunk egy raktárkezelő programot és azt mondjuk neki, hogy figyelj Mariska néni, sütsz nekem két tálca süteményt, megcsinálom a programodat, de vegyél 15 millióért egy Oracle-t alá. Nem fogja kérni. Nagyon fontos, hogy az SQL szerver kiválasztásakor a feladat méretéhez legjobban illő verziót keressük meg.

Az SQL szerverek árát jelentősen befolyásolja és emeli a hozzájuk adott kiegészítő szoftver és menedzser felület. Természetesen jelentős és sokszor nélkülözhetetlen a terméktámogatást is kapunk a pénzünkért.

Tehát ezek fizetősek; a pénzért adnak szolgáltatást, supportot. Hogyha egy éles rendszert csinálunk egy nagy cégnek, akkor ez fontos.

A másik csoport, az ingyenes szoftverek csoportja.

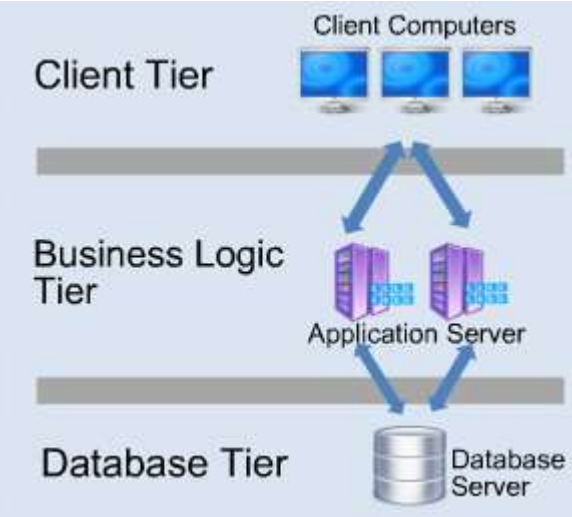
Ide szoktam sorolni a MySQL-t is. Az 5.1-es verziótól kezdve tud tárolt eljárásokat kezelni (ez egy nagyon jó, nagyon hiányzó dolog volt). Tehát a MySQL-el nekem csak ennyi volt a problémám, hogy nem tud tranzakciót kezelni, meg még egyéb ilyen apróságokat nem teljesít, amiket még kellene; tehát a bankok valószínűleg még egy darabig nem fogják használni, mert ATM pénzfelvételre nem lesz alkalmas, nem tudja lekezelni. De már majdnem ingyenes.

Másik lehetőség a PostgreSQL. A PostgreSQL is régóta tudja a tárolt eljárásokat, szépen kezeli a triggereket, tudja kiszolgálni a tranzakciókat (nem csak autotranzakciók vannak benne), mégis kevésbé terjedt el, mint a MySQL, pedig teljesen jó és ingyenes rendszer;ajánlom esetleg majd szemrevételezésre.

Van még egy nagyon érdekes rendszer, amit úgy hívnak , hogy Firebird. Az Interbassel egyenértékű, vele 100%-ban kompatibilis SQL szerver.

A Firebird kiválóan alkalmas egy kis, esetleg egy közepes vállalkozás adattárolási és nyilvántartási rendszerek kezelésére.

Ahol komoly adatreplikáció van, ott már tényleg nem alkalmas. És ezeknek a rendszereknek, pl. a Firebird-nek megvan az az előnye, hogyha írtunk egy rendszert és szeretnénk eladni kis- és középvállalkozásoknak, akkor ezek kiszolgálják. Ugyanúgy el tudjuk adni változtatások nélkül lokális rendszerben is, tehát amikor Mariska néni nyit egy virágboltot és mondja hogy számláznia kell, vagy számlát kell kiállítani esetleg, mondjuk egy héten ötöt, meg bevételezni kell egy héten kétszer, azt is kiválóan kiszolgálja. Nem kell neki 5 processzoros gép 100 gigabájttal, hanem egy sima laptopon elmegy. A Firebird telepítőjének még menedzser-felülete sincsen, tehát hat vagy hét megabájttal megy, a tranzakció-kezelés kiváló benne, tárolt eljárások, triggerek, hatos verziótól már simán megvannak. Gyakorlatilag mindent tud; csak nem monumentálisban, hanem ilyen kicsi vagy közepes vállalkozás szinten. Ajánlom az olyanok figyelmébe, akik egy kicsit vájt fülűek az ilyenekre.



Multi-Tier

Többrétegű architektúra, itt már nem feltétlenül hardverre gondolunk, hanem logikai rétegekre. Van egy SQL szerver és vannak a kliens programok. Ugye ez a kliens – szerver architektúra. Egy-két köztes réteget ezek közé azzal a feltétellel, hogy ezt a réteget kereshetik meg a kéréssel a kliensek, és tőle várják a választ. És csak ez a réteg nyúlhat az SQL szerverhez, csak ez a réteg fordulhat az SQL rendszerhez kérdéssel. A kliens program nem nyúlhat az SQL szerverhez közvetlenül. A kliens-szerver architektúrában a kliens program közvetlenül az SQL szervert éri el. Ott az SQL szerver által menedzselt adatbázisban tárolt eljárást hívom meg. Itt nem. Itt van egy köztes réteg, úgy szokták hívni, hogy üzleti logika. Egy eljárás, függvény, metódus gyűjtemény, amit a kliensek hívogathatnak. A BL(Bussnis Logic Layer) felel azért, hogy az SQL rendszerrel kommunikáljanak. A BL-t nem lehet megkerülni. Onnan fejlődött ki jellemzően, hogy milyen kellemes az, ha egy programot nem kell telepíteni a kliensre, hanem azt mondja a kliens, hogy nekem úgy is van egy böngészőm, beírunk egy webcímet, és akkor majd úgy kommunikálok. A webhelyen meg fog jelenni valamilyen adat. Nyilván azért ez szélsőséges, mert nagyon sok olyan rendszert tudunk mondani, amit nem tud kiszolgálni egy böngésző. Ez komoly rendszereknél biztos, hogy kettő hardver eszköz, kettő szerver. Tehát az nem komoly rendszer, amikor azt mondják, hogy egy webszervert és egy adatbázis-szervert egy vasra tesznek. Ez nem rendszer adatbiztonság szempontjából. A biztonságos dolgok miatt szoktuk azt mondani, hogy az egyik vas egy adatbázis szerver, amit egy úgynevezett DMZ-ben helyezünk el (demilitarizált zóna) sok tűzfallal körbebástyázva, bezárva. Egyszerűen arról van szó, hogy ezek az adatok értékek; tehát egy cégnek ezek vesznek el vagy szivárognak ki, akkor óriási károkat okozhatnak.

Egy nagyon egyszerű példa: amikor csináltuk az EGERFOOD rendszert (http://www.ektf.hu/ret/fo_profil/, illetve http://egerfood.eu/), van benne hat cég egy – egy termékével, élelmiszer-gyártó cégek. Halmajugrán van a Detki keksz gyára, ők a legegyszerűbb termékkel léptek be az EGERFOOD rendszerbe, úgy hívták, háztartási keksz. Mikor elmentünk a céghez megbeszélni, hogy milyen rendszert fogunk létrehozni, hogyan lesznek az adatkapcsolatok, az első és legfontosabb kérdés az adatbiztonság volt. Leültettek, bejött a főnökasszony, még három perce sem beszéltünk, hogy mit szeretnénk, amikor azt mondta, hogy fiúk, álljunk meg, azt mondják meg, hogyan lehet biztosítani, hogy a recept és az adatok, amit használunk és küldözgetünk az interneten keresztül Halmajugra és a főiskola között, nem kerülnek másnak a kezébe? Bemutattuk nekik, hogy VPN (Virtual Privat Network), meg hogy a WCF (Windows Communication Foundation) titkosítási rendszerét használjuk, meg különben is AS 128-al kódolunk már röptében mindent. Egy három rétegű biztonsági rendszert felvázoltunk nekik, szépen felrajzoltuk. Különben ez is valósult meg, tehát nem a levegőbe beszéltünk. De a terv akkor még terv volt. Azt mondta, hogy jó, köszöni, elfogadható, mehetünk. Az iparban szélsőségesen megkövetelik a fejlesztőktől az adatbiztonságot. Nyilván amikor arra kerül a sor, hogy egymillióval többe kerül, akkor „húzzák a szájukat”, de ez természetes, hiszen be kell fektetni. Úgyhogy tényleg az van mostanában, hogy adatbázis-szerver – demilitarizált zóna és üzleti logika egy külön számítógép; ha webes a felépítmény, akkor az egy másik számítógép. A hozzáférés megvalósulhat pda-n, mobiltelefonon, laptopon, bárhogy.

Vastag kliens

A vastag kliens (szokták még a kövér kliens vagy gazdag kliens nevet is használni) képes arra, hogy önmaga hajtson végre nagyobb adatmennyiségekkel feldolgozásokat, amikor a szerver inkább elsődleges tárolóként viselkedik. Ennek ellenére, a kifejezés inkább a számítógép szoftverére vonatkozik, és egyre inkább alkalmazzák hálózati számítógépek esetén, ahol a számítógép jelentős hálózati alkalmazásokat (is) futtat.

Vékony kliens

A vékony kliens (angol terminológiával: thin client) egy minimális eszközökkel rendelkező kliens. Ez a kliens típus a szükséges erőforrásokat is a távoli (host) gépen veszi igénybe. Egy vékony kliens feladata többnyire kimerül az alkalmazás szerver által küldött adatok grafikus

megjelenítésében; a tényleges, nagy mennyiségű adat mozgatását, kezelését igénylő feladatot az alkalmazás szerver végzi el.

Alapvető szerkezetek

Séma: minden adatbázisnak van egy belső struktúrája, ami tartalmazza az összes adatelem és a közöttük lévő kapcsolatok leírását. Ezt a struktúrát az adatbázis sémájának nevezzük.
A legjelentősebb **metaadatok** az adatok típusának definícióját, az egyes adatok azonosítására szolgáló neveket tartalmazzák, utalásokat arra, hogy milyen kapcsolatok, összefüggések vannak az adatok között, valamint az adatbázis adminisztrációjával kapcsolatos információkat. Azaz segítségükkel tudjuk tárolni a tényleges adatok melletti strukturális információt.
Az adatbázis felépítése az alkalmazott modelltől függően más és más lehet. Van azonban néhány olyan általános elv, melyet szinte minden adatbázison alapuló alkalmazásnál használnak. Ezek:
A **tábla**, vagy adattábla egy kétdimenziós tábla, mely a logikailag szorosan összetartozó adatokat szemlélteti. A tábla oszlopokból és sorokból áll.
A **rekord**, az adatbázis egy sora. Egy rekordban tároljuk az egymással összefüggő adatokat. A tábla sorai tartalmazzák az egyes tulajdonságok konkrét értékeit.
A **mező**, a tábla egy oszlopa. Minden egyes oszlop az adott dolog egy jellemzőjét jelenti, mely névvel és típussal van ellátva.
Az elemi adatok, a tábla celláiban szereplő értékek, melyek az egyed konkrét tulajdonságai.
Az **egyed** az, amit le akarunk írni, amelynek az adatait tároljuk és gyűjtjük az adatbázisban. Az egyedet idegen szóval entitásnak nevezzük. Egyednek tekintünk például egy személyt.
Egyednek nevezzük azokat a dolgokat, objektumokat, amelyek egymástól jól elkülöníthetők, melyekről adatokat tárolunk és tulajdonságokkal jellemzünk. Egyedek lehetnek például a dolgozó, kifizetés, anyag, személy, stb. Ebben a formában az egyed mint absztrakt fogalom szerepel. Mondhatjuk azt is, hogy az egyed konkrét dolgok absztrakciója. Az absztrakt egyedekre szokás használni az egyedtípus kifejezést is.
Az **attribútum**, a tulajdonság, az egyed valamely jellemzője. Az egyed az attribútumok összességével jellemezhető. Egy személy egy jellemzője lehet például a neve.

Az egyed típus az egyedre vonatkozóan megadott tulajdonságok összessége. Például egy személy leírható a nevével, a születési dátumával, testmagasságával, haja és szeme színével együttesen.

Az **egyed előfordulás** az egyedre megadott konkrét tulajdonságok. Például Koltai Lea Kiara 5 éves, barna hajú, barna szemű, 110 cm magas, óvodás. Az egyed előfordulások a rekordoknak felelnek meg. A gyakorlatban az egyedtípust szokás rekordtípusnak is nevezni. (rekord- vagy struktúratípus).

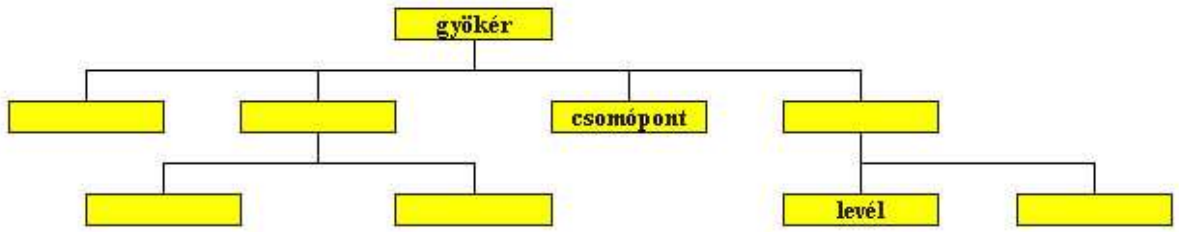
Adatredundancia, amikor egy adatot egynél több helyen tárolunk egy számítógépes rendszerben, adatredundanciáról beszélünk. Mivel szinte lehetetlen elkerülni a redundanciát, arra kell törekednünk, hogy a többszörös előfordulásokat minimálisra redukáljuk. Ennek módszere, hogy az ismétlődő adatokat az adatbázis tervezése során kiemeljük, és külön tároljuk, a megfelelő helyen hivatkozva rá.

Adatmodellek fejlődése

Modell készítése a tudományos életben gyakori módszer a lényegmegismerésre. Az informatikában azokat a modelleket nevezzük adatmodelleknek, amelyek az adatok szerkezetének leírására szolgálnak.
Az adatbázis-kezelés során többféle adatmodell alakult ki, ezek közül három terjedt el igazán. Azonban meg kell említeni, hogy kialakulóban van egy új adatmodell az új programozási technikáknak köszönhetően, az objektum orientált modell.

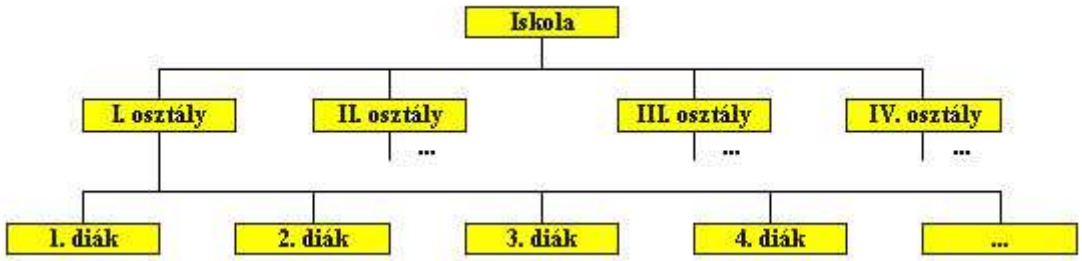
Hierarchikus

Ez a legősibb adatmodell. Az adatokat egy hierarchikus szerkezetben tárolja, amely egy fához hasonlítható. A fa mindegyik csomópontja egy rekordtípusnak felel meg. Az adatok között un. szülő-gyermek kapcsolat van. Minden adatnak tetszőleges számú leszármazottja lehet, de csak egy őse. Ez a modell leginkább egy-egy és egy több jellegű kapcsolatok megvalósítására alkalmazható. Napjainkra ezt a modellt teljesen kiszorította a relációs adatmodell.



Az adatbázis több egymástól független fából állhat. A fa csomópontjaiban és leveleiben helyezkednek el az adatok. A közöttük levő kapcsolat, szülő gyermek kapcsolatnak felel meg. Így csak 1:n típusú kapcsolatok képezhetők le segítségével. Az 1:n kapcsolat azt jelenti, hogy az adatszerkezet egyik típusú adata a hierarchiában alatta elhelyezkedő egy vagy több más adattal áll kapcsolatban.
A hierarchikus modell természetéből adódóan nem ábrázolhatunk benne n:m típusú kapcsolatokat (lásd a háló modellt). Emellett további hátránya, hogy az adatok elérése csak egyféle sorrendben lehetséges, a tárolt hierarchiának megfelelő sorrendben.

A hierarchikus adatmodell alkalmazására a legkézenfekvőbb példa a családfa. De a főnök-beosztott viszonyok vagy egy iskola szerkezete is leírható ebben a modellben. Az iskola esetén többféle hierarchia is felépíthető. Egyrészt az iskola több osztályra bomlik és az osztályok tanulókból állnak. Másrészt az iskolát az igazgató vezeti, a többi tanár az ô beosztottja és a tanárok egy vagy több tantárgyat tanítanak.



Iskola hierarchikus felépítése a diákok szemszögéből

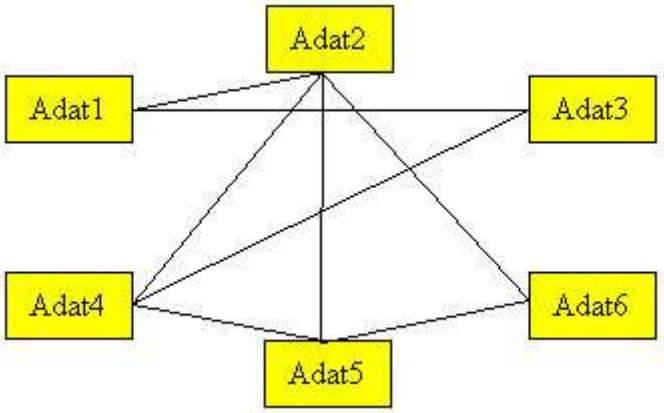
ábra Iskola hierarchikus felépítése a tanárok szemszögéből

Hálós

Ez a modell a hierarchikus modell továbbfejlesztése. A különbség abban mutatkozik meg, hogy míg az előbbi modell gráfja csak fa lehet, addig ebben a modellben tetszőleges gráf előfordulhat. Tehát egy egyedtípusnak több őse is lehet, az adatok között tetszőleges kapcsolatrendszer alakítható ki. A hálós adatmodellben kezelhetők a több-több kapcsolatok is. Hátránya az, hogy nagy a tároló igénye. Nagy-gépes környezetben fordul elő. Napjainkra ez a modell elavult.

A hálós adatmodell esetén az egyes azonos vagy különböző összetételű adategységek (rekordok) között a kapcsolat egy gráffal írható le. A gráf csomópontok és ezeket összekötő élek rendszere, melyben tetszőleges két csomópont között akkor van adatkapcsolat, ha őket él köti össze egymással. Egy csomópontból tetszőleges számú él indulhat ki, de egy él csak két csomópontot köthet össze. Azaz minden adategység tetszőleges más adategységekkel lehet kapcsolatban. ebben a modellben n:m típusú adatkapcsolatok is leírhatók az 1:n típusúak

mellett. A hierarchikus és a hálós modell esetén az adatbázisba fixen beépített kapcsolatok következtében csak a tárolt kapcsolatok segítségével bejárható adat-visszakeresések oldhatók meg hatékonyan (sok esetben hatékonyabban, mint más modellekben). További hátrányuk, hogy szerkezetük merev, módosításuk nehézkes.



Hálós adatmodell

Relációs

A relációs adatmodell kidolgozása Codd nevéhez fűződik (1971). Azóta fontos szerepet játszik az adatbázis kezelők alkalmazásában. A relációs modell előnyei a következők:

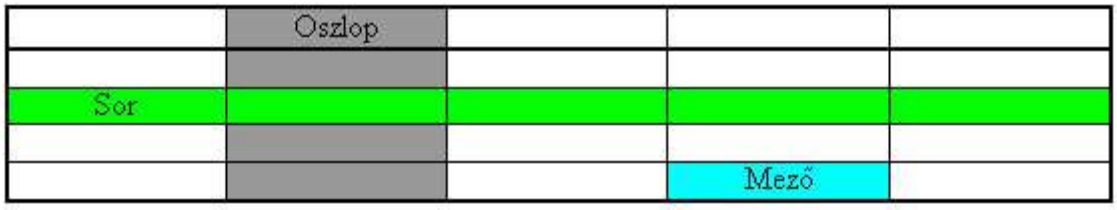
- A relációs adatszerkezet egyszerűen értelmezhető a felhasználók és az alkalmazás készítők számára is, így ez lehet közöttük a kommunikáció eszköze.
- A logikai adatmodell relációi egy relációs adatbázis kezelő rendszerbe módosítások nélkül átvihetők.

A relációs modellben az adatbázis tervezés a normál formák bevezetésével egzakt módon elvégezhető

A relációs adatmodell jellemzője, hogy az adatokat több, egymással összekapcsolt rendszerben ábrázolja. Manapság ez a legelterjedtebb adatmodell. Alapját a matematikában is használatos reláció jelenti. Egy új módszert alkalmaz az adatbázis lekérdezések megvalósítására a relációkon értelmezett műveletek segítségével. Az SQL (Structured Query Language) Strukturált lekérdező nyelv egy komplex adatbázis-lekérdező nyelv, mellyel megvalósíthatjuk a lekérdezéseket és különböző adatbázis-kezelő műveleteket. Az Access a relációs adatmodellt használja, ezért bővebb részletezést igényel.

Ebben a modellben az adatokat egy kétdimenziós táblában elrendezve ábrázoljuk, melyben az adatok egymással logikai kapcsolatban állnak.

A reláció nem más mint egy táblázat, a táblázat soraiban tárolt adatokkal együtt. A relációs adatbázis pedig relációk és csak relációk összessége. Az egyes relációkat egyedi névvel látjuk el. A relációk oszlopaiban azonos mennyiségre vonatkozó adatok jelennek meg. Az oszlopok névvel rendelkeznek, melyeknek a reláción belül egyedieknek kell lenniük, de más relációk tartalmazhatnak azonos nevű oszlopokat. A reláció soraiban tároljuk a logikailag összetartozó adatokat. A reláció sorainak sorrendje közömbös, de nem tartalmazhat két azonos adatokkal kitöltött sort. Egy sor és oszlop metszésében található táblázat elemet mezőnek nevezzük, a mezők tartalmazzák az adatokat. A mezőkben oszloponként különböző típusú (numerikus, szöveges stb.) mennyiségek tárolhatók. A reláció helyett sokszor a tábla vagy táblázat, a sor helyett a rekord, az oszlop helyett pedig az attribútum elnevezés is használatos.



Például egy személyi adatokat tartalmazó reláció a következő lehet:

Személy

Személyi szám	**Név**	**Város**	**Foglalkozás**
1 650410 1256	Kiss László	Győr	kőműves
2 781117 0131	Nagy Ágnes	Szeged	tanuló
1 610105 1167	Kiss László	Budapest	lakatos

Az előző relációból a személyi szám oszlopot elhagyva relációnak tekinthető-e a táblázat? Mivel nem zárható ki, hogy két azonos nevű és szakmájú személy éljen egy településen belül a személyi szám nélkül két azonos sor is szerepelhetne, mely a relációban nem megengedett.

Anyag		
kód	készlet	egységár
1206	389	274
967	2012	65

A reláció oszlopainak

12	654	712

elnevezésére célszerű a tartalomra utaló elnevezést használni még akkor is, ha ez esetleg több gépeléssel is jár. Ehhez álljon itt a következő példa:

R		
A	B	C
1206	389	274
967	2012	65
12	654	712

Az előző két reláció ugyanazokat az oszlopokat tartalmazza, de a bal oldali esetben további feljegyzésekre van szükség az egyes oszlopok tartalmának leírására.

A relációktól általában megköveteljük, hogy ne tartalmazzanak más adatokból levezethető vagy kiszámítható információkat. Például az anyag relációban (2.3 ábra) fölösleges lenne egy érték oszlopot is tárolni, mivel ez az adat a készlet és az egységár szorzataként kiszámítható a rendelkezésre álló adatokból. Hasonlóképpen a személyi szám mellett nincs értelme külön a születési dátumot nyilvántartani, mert az része a személyi számnak, abból előállítható.

A táblázattal kapcsolatos alapkövetelmények:

- Minden táblázat egyértelmű azonosítóval bír.
- A sorok és oszlopok metszéspontjában található adatok egyértékűek, ezeket nevezi elemi adatmezőknek.
- Az oszlopokban lévő adatok azonos jellegűek.
- Minden oszlopnak egyedi neve van.
- Ugyan annyi adat található a táblázat minden sorában.
- Nem lehet két azonos sor a táblázatban.
- A sorok és oszlopok sorrendje tetszőleges.

KULCS. Fontos szerepe van azoknak a tulajdonságoknak, amelynek értékei a többi tulajdonság értékeit egyértelműen meghatározzák. Ez azt jelenti, hogy ha az ilyen tulajdonságok értékeit megadjuk, akkor az egyértelműen definiál egy előfordulást. Azokat a tulajdonságokat, amelyek egyértelműen meghatározzák az egyedtípus egy elemét, kulcsnak nevezzük. A kulcsok fontos szerepet töltenek be az adatmodell kialakításánál. A tervezés

során általában meg szokták adni, mely attribútumok fogják a kulcsokat alkotni. Elvileg egy egyednek több kulcsa is lehet, de a legtöbb esetben egyet szokás kiválasztani, amely a leginkább alkalmas az egyértelmű azonosításra. Ezt hívjuk elsődleges kulcsnak.

Kulcsjellegű tulajdonság mindig található. Ha a tényleges adatok között nem lenne ilyen, akkor bevezethetünk egy olyan tulajdonságot, amelynek értékei sorszámok, kódszámok, speciális azonosítók. Ez betöltheti az elsődleges kulcs szerepét.

Láthatjuk hogy az azonosítók, kódszámok szinte minden alkalmazásnál előfordulnak. A számítógép jellegéből adódóan ezek alkalmasak az előfordulások pontos meghatározására. Bizonyos esetekben a laikus felhasználó számára nehéz¬séget okozhat, hogy a kódnál már egy apró elírás is egész más eredményt szolgáltathat. Aki számítógéppel dolgozik, annak tudomásul kell venni a kódok használatát, és pontos munkára kell törekednie.

Kapcsolat

Az adatmodell harmadik fontos elemét a kapcsolatok jelentik. Kapcsolatnak nevezzük az egyedek közötti összefüggést, viszonyt. Például a már jól ismert bérszámfejtő rendszerben a dolgozó és a kifizetés egyedek között létezik egy természetes kapcsolat. Ez azt mondja meg, hogy az egyes dolgozókhoz mely kifizetések tartoznak.

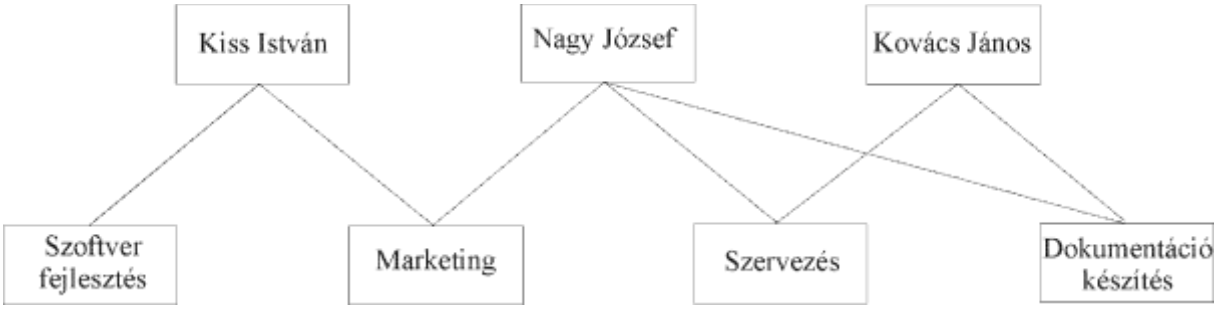
A fentiek alapján kapcsolatok az egyedhalmazok elemei között alakíthatók ki. Osztályozhatjuk a kapcsolatokat aszerint, hogy egy-egy elemhez hány másik elem tartozik. Ennek a kapcsolat számítógépes reprezentációja szempontjából van jelentősége. Sokkal egyszerűbb ugyanis egy olyan kapcsolatot megvalósítani, ahol egy egyed előforduláshoz csak egyetlen másik egyed előfordulás tartozhat, mint ha több. Az előbbinél elegendő lehet egy mutató, míg az utóbbinál valamilyen összetett adatstruktúrára van szükség, például halmazra, vagy listára. A kapcsolatokat ezek alapján három csoportba szokás sorolni:

- Egy-egy típusú
- Egy-sok típusú
- Sok-sok típusú

Egy–egy típusú kapcsolat esetén az egyik egyed minden egyes előfordulásának a másik egyed pontosan egy előfordulása tartozik. Egy–egy típusú kapcsolat például a férfi és a nő egyedek között a házastárs kapcsolat.

A következő csoportot az egy-sok típusú kapcsolatok alkotják. Ezeknél az egyik egyed minden előfordulásához a másik egyed több előfordulása tartozhat. Például a bérszámfejtő rendszerben egy-sok kapcsolat van a dolgozók és a kifizetések között. A kapcsolat alapja az, hogy melyik dolgozóhoz mely kifizetések tartoznak. Világos, hogy egy dolgozóhoz több kifizetés tartozhat, viszont egy kifizetés mindenképpen csak egyetlen dolgozóhoz kapcsolódik.

A kapcsolatok legáltalánosabb formáját a sok-sok kapcsolatok jelentik. Sok-sok kapcsolat esetén mindkét egyed előfordulásaihoz a másik egyed több előfordulása tartozhat. Tegyük fel hogy dolgozói rendszerünkben azt is nyilvántartjuk, hogy melyik dolgozó milyen témákon dolgozik. Egy dolgozó több témában is tevékenykedhet és egy témán több dolgozó dolgozhat. Ebben az esetben tehát sok-sok kapcsolatról van szó. Ezt a következő ábra ezt szemlélteti.



A sok-sok kapcsolatok láthatóan egy-sok kapcsolatokon alapszanak. Ha bármelyik egyed szempontjából nézzük, akkor egy-sok kapcsolatot fedezhetünk fel. Ezért minden sok-sok kapcsolat felbontható két egy-sok kapcsolatra.

Az eddigiekben olyan kapcsolatokról beszéltünk, amelyek két egyed között létesíthetők. Ezek az úgynevezett bináris kapcsolatok.

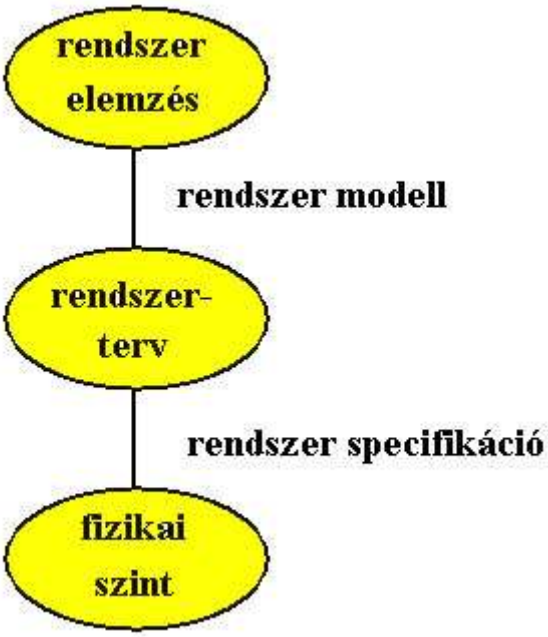
Adatbázis tervezés, és eszközei

Az adatbázis megtervezéséhez elsősorban tudnunk kell, hogy milyen adatbázis-kezelő programot használunk. A relációs adatbázis-kezelőnél, mint a relációs adatbázis-kezelőknél rendszerint az adatokat csoportosítjuk, és az egymással szoros összefüggésben lévő adatokat ugyanabban a táblában tároljuk, majd meghatározzuk, hogy az így kapott táblák milyen kapcsolatban vannak egymással. Tehát egy adatbázis megtervezése, létrehozása igen összetett feladat, s igényel némi kreativitást. Az adatbázisokat úgy kell megtervezni, hogy eleget tegyenek bizonyos kritériumnak, például, hogy minimális legyen bennük az adatredundancia, teljesüljenek a különböző adatfüggetlenségek, stb. Nincs általános tervezési technika, mely alkalmazható lenne minden adatbázis elkészítéséhez, de van menete, melyet érdemes követni.
Mindenekelőtt, meg kell határoznunk a célokat, melyek szoros kapcsolatban vannak a felhasználó igényeivel. A tervezésnél fontos, hogy a tervezést végző személy kellő ismeretekkel rendelkezzen az adott rendszer felhasználóinak szakterületén. Ez az információigény meghatározásának szakasza, amely során az adatok, formátumok, algoritmusok kialakítását is el kell végezni. Jellemző, hogy a tervezést általában nem a programozó végzi, hanem a szervező, aki jártas a tervezésben, mely során megismeri a pontos felhasználói igényeket. Alapos kutatást végez, segítségére lehetnek különböző jelentések, dokumentumok, melyek forrásként szolgálhatnak a rendszer megtervezéséhez.
Miután mindezeket megvalósítottuk, kezdődhet a logikai adatbázis-tervezés. Ebben a szakaszban főként az adatok és a közöttük lévő kapcsolatokra helyeződik a hangsúly. Itt történik az egyedek meghatározása, megadásra kerülnek az egyedeket leíró tulajdonságok, valamint feltérképezésre kerülnek az egyedek közötti kapcsolatok, mindemellett ügyelve az adatredundancia minimalizálására.
A fizikai tervezés szakasza jobban elkülönül az első két szakasztól, ahol is az adatbázisok számítógépen történő létrehozása valósul meg a korábbi tervek alapján. Elkészül a prototípus, a rendszer első verziója, ami még nem teljes, változtatásra, tökéletesítésre szorul.

Az adatbázis tervezés főbb lépései

Az adatbázis tervezés egy folyamat, mely több lépésből tevődik össze. Először az adatbázisban leképezendő rendszert elemzésnek vetjük alá és meghatározzuk a tárolandó

adatok körét, azok egymás közötti kapcsolatait és az adatbázissal szemben felmerülő igényeket. Ezután következik a rendszer tervezés, melynek eredménye az adatbázis logikai modellje. Végül fizikai szinten képezzük le a logikai adatbázis modellt a felhasználható szoftver és hardver függvényében.



1. Követelményelemzés: először is meg kell határoznunk az adatbázis célját. Kutatást kell végeznünk a rendszer megtervezéséhez. Át kell gondolnunk, milyen információkhoz szeretnénk jutni az adatbázisból és, hogy melyek azok az adatok amelyeket tárolnunk kell az egyedről.
2. Egyedek, táblák meghatározása: az összegyűjtött adatokat rendszerezésük után információrendszerbe kell szervezni. Ez az információrendszer egyedekkel foglalkozik. Az egyedek tárolása fizikailag egy táblában történik, melynek soraiba (rekordjaiba) kerülnek az egyedpéldányok, a rekord mezőibe (oszlopokba) pedig az attribútumok. Fontos, hogy minden adatot csak egy táblában tároljunk, azért, hogy a későbbi módosításkor csak egy helyen kelljen frissíteni az adatokat. Egy táblában csak egy adott témára vonatkozó információt tárolunk.
3. Mezők, attribútumok meghatározása: ez a konkrét tervezés szakasza, itt tervezzük meg a táblákat, valamint meghatározzuk a táblákat felépítő mezőket. Az attribútumokat osztályozhatjuk az alábbiak szerint:

a. egyszerű, azaz tovább nem bontható, valamint összetett. Az összetett attribútum több egyszerű értékből áll.

b. egyértékű: minden előfordulásnál csak egy értéket vehet fel. A többértékű minden előfordulásnál több értéket is felvehet.

c. a tárolt attribútum értékeit az adatbázis tárolja. A származtatott értéke más attribútumok alapján határozható meg.

4. Azonosítók meghatározása: fontos, hogy a táblákban tárolt adatokat egyértelműen kell azonosítani. Elsődleges kulcsra minden olyan táblában szükség van, amelynek rekordjait egyenként szeretnénk azonosítani. Az elsődleges kulcs olyan azonosító, amelynek értékei nem ismétlődhetnek az adott táblában. Az elsődleges kulcsnak fontos szerepe van a relációs adatbázisokban. Segítségével növelhetjük a hatékonyságot, gyorsítja a keresést és az adatok összegyűjtését.

Háromféle elsődleges kulcs alkalmazható:

a. számláló típusú: ez a legegyszerűbb elsődleges kulcs. Ilyenkor létre kell hozni egy számláló típusú mezőt. Az Access minden egyes új rekord számára egyedi sorszámot generál.

b. egy mezőből álló elsődleges kulcs: a kulcs nem számláló típusú, ha nem tartalmaz egyetlen ismétlődő értéket sem (például adószám).

c. több mezőből álló elsődleges kulcs: ilyen kulcsot több mező felhasználásával képezünk. Erre akkor kerül sor, ha egyetlen mező egyediségét sem tudjuk biztosítani.

5. Kapcsolatok meghatározása: a táblák rekordjait kapcsoljuk össze az elsődleges kulcsmezők segítségével. A kapcsolat 2 egyed összetartozását jelenti.

A kapcsolat számosságát három csoportba oszthatjuk: (ezt a későbbiekben tárgyaljuk bővebben)

a. Egy az egyhez kapcsolat.

b. Egy a többhöz kapcsolat.

c. Több a többhöz kapcsolat.

6. Ellenőrzés: A mezők, táblák és kapcsolatok megtervezése után meg kell nézni a tervet, hogy nem maradt-e benne hiba. Közvetlenül az ellenőrzés után könnyebb az adatbázis tervét módosítani, mint amikor már fel van töltve adatokkal.

7. Adatbevitel: miután elvégeztük a szükséges javításokat és ellenőrzéseket, bevihetjük az adatokat a már létező táblákba. Továbbá kialakíthatjuk a többi objektumot. Van lehetőség űrlapok, jelentések és lekérdezések készítésére (ezekről később részletesen).

Normalizálás

A relációs adatbázis felépítésének alapja a normalizálás, amely az adatok optimális elhelyezési módját megadó módszert jelenti. Nem megfelelően felépített adatbázis esetén az adatszerkezetben különböző ellentmondások, anomáliák keletkezhetnek. A normalizálás lehetővé teszi az adatok megfelelő strukturálását, valamint elősegíti az anomáliák kiküszöbölését és a redundancia csökkentését.

Anomáliák:

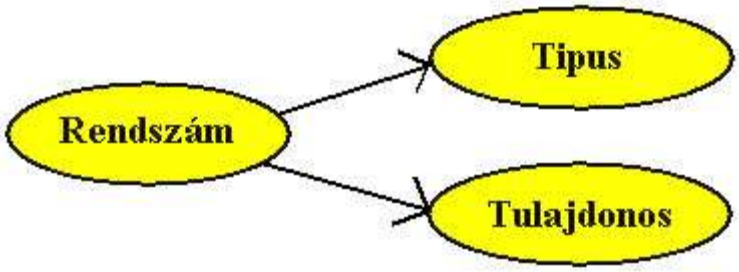
- Beszúrási anomália: egy rekord felvétele egy másik, hozzá logikailag nem kapcsolódó adat beszúrását kívánja meg.
- Törlési anomália: az elem törlésekor a nem hozzá tartozó adatcsoportot is elveszítjük.
- Módosítási anomália: egy adat megváltoztatása miatt, az adat összes előfordulási helyén el kell végezni a módosításokat az adatbázisban.

Függőségek

Funkcionális függőség: ha egy rendszerben szereplő egyik tulajdonságtípus bármely értékéhez egy másik tulajdonságtípusnak csakis egy értéke rendelhető hozzá.

Például: egy személyi számhoz csak egy név tartozhat, de ugyanahhoz a névhez több személyi adatazonosító szám is kapcsolódhat. Egy a többhöz kapcsolat.

A funkcionális függõség jobb oldalán több attribútum is állhat. Például az **AUTÓ_RENDSZÁM -> TIPUS, TULAJDONOS** funkcionális függõség azt fejezi ki, hogy az autó rendszámából következik a tipusa és a tulajdonos neve, mivel minden autónak különbözõ a rendszáma, minden autónak egy tulajdonosa és tipusa van. Ezt diagrammal is ábrázolhatjuk.

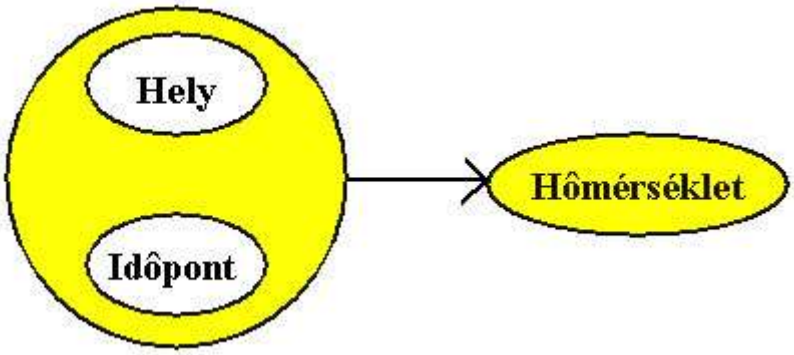


Kölcsönös funkcionális függőség: ha az előbbi feltétel mindkét irányba igaz. Például: rendszám – motorszám. Egy az egyhez kapcsolat.

Az is előfordulhat, hogy két attribútum kölcsönösen függ egymástól. Ez a helyzet például a házastársak esetén FÉRJ_SZEM_SZÁMA -> FELESÉG_SZEM_SZÁMA FELESÉG_SZEM_SZÁMA <- FÉRJ_SZEM_SZÁMA. Mindkét funkcionális kapcsolat igaz és ezt a FÉRJ_SZEM_SZÁMA <-> FELESÉG_SZEM_SZÁMA jelöléssel fejezzük ki. Természetesen a fenti összefüggés a többnejűséget megengedő országokban nem teljesül.

A funkcionális függőség bal oldalán több attribútum is megjelenhet, melyek együttesen határozzák meg a jobb oldalon szereplő attribútum értékét. Például hőmérsékletet mérünk különböző helyeken és időben úgy, hogy a helyszínek között azonosak is lehetnek. Ebben az esetben a következő funkcionális függőség áll fenn az attribútumok között:

HELY, IDŐPONT -> HŐMÉRSÉKLET. A fenti összefüggést az alábbi diagrammal is jelölhetjük:



Funkcionálisan függetlenek: ha ez előbbi viszony a két tulajdonságtípus között nem áll fenn. Például: tanuló szemének a színe és az iskola helye.

Tranzitív funkcionális függőség: ha az egyedtípuson belül egy leíró tulajdonságtípus konkrét értékei meghatároznak más leíró tulajdonság értékeit.

Reláció kulcs

A reláció kulcs a reláció egy sorát azonosítja egyértelműen. A reláció - definíció szerint- nem tartalmazhat két azonos sort, ezért minden relációban létezik kulcs. A reláció kulcsnak a következő feltételeket kell teljesítenie

- az attribútumok egy olyan csoportja, melyek csak egy sort azonosítanak (egyértelműség)
- a kulcsban szereplő attribútumok egyetlen részhalmaza sem alkot kulcsot

- a kulcsban szereplő attribútumok értéke nem lehet definiálatlan (NULL)

A definiálatlan (NULL) értékek tárolását a relációs adatbázis kezelők speciálisan oldják meg. Numerikus értékek esetén a NULL érték és a 0 nem azonos.

Egy relációban tartsuk nyilván az osztály tanulóinak személyi adatait

Személyi szám	**Születési év**	**Név**

SZEMÉLY_ADATOK=({ SZEMÉLYI_SZÁM, SZÜL_ÉV, NÉV}).

A SZEMÉLYI_ADATOK relációban a SZEMÉLYI_SZÁM attribútum kulcs, mert nem lehet az adatok között két különböző személy azonos személyi számmal. A születési év vagy a név nem azonosítja egyértelműen a reláció egy sorát mivel ugyanazon a napon is született tanulók vagy azonos nevűek is lehetnek az osztályban. Vajon a személyi szám és a születési év kulcsa-e a személyi adatok relációnak? Együtt a reláció egy sorát azonosítják, de nem tesznek eleget a kulcsokra vonatkozó azon feltételnek, hogy a bennük szereplő attribútumok részhalmaza nem lehet kulcs. Ebben az esetben a személyi szám már kulcs, így bármelyik másik attribútummal kombinálva már nem alkothat kulcsot.

Előfordulnak olyan relációk is, melyekben a kulcs több attribútum érték összekapcsolásával állítható elő. Készítsünk nyilvántartást a diákok különböző tantárgyakból szerzett osztályzatairól az alábbi relációval:

NAPLÓ=({SZEMÉLYI_SZÁM, TANTÁRGY, DÁTUM, OSZTÁLYZAT)}

Személyi szám	**Tantárgy**	**Dátum**	**Osztályzat**

A NAPLÓ relációban a SZEMÉLYI_SZÁM nem azonosít egy sort, mivel egy diáknak több osztályzata is lehet akár ugyanabból a tantárgyból is. Ezért még a SZEMÉLYI_SZÁM és a TANTÁRGY sem alkot kulcsot. A SZEMÉLYI_SZÁM, TANTÁRGY és a DÁTUM is csak akkor alkot kulcsot, ha kizárjuk annak lehetőségét, hogy ugyanazon a napon ugyanabból a tantárgyból egy diák két osztályzatot kaphat. Abban az esetben, ha ez a feltételezés nem

tartható, akkor nem csak az osztályzat megszerzésének dátumát, hanem annak időpontját is tárolni kell. Ilyenkor természetesen a NAPLÓ relációt ezzel az új oszloppal ki kell bővíteni.

Nem csak összetett kulcsok fordulhatnak elő a relációkban, léteznek olyan relációk is, melyekben nem csak egy, hanem több kulcs is található. Ennek illusztrálására nézzük meg a következő relációt

KONZULTÁCIÓ=({TANÁR, IDőPONT, DIÁK)}

Tanár	**Időpont**	**Diák**

Reláció több kulccsal

A KONZULTÁCIÓ relációban a tanár illetve a diák oszlopban olyan azonosítót képzelünk, mely a személyt egyértelműen azonosítja (például személyi szám). Minden egyes diák több konzultáción vehet rész, minden tanár több konzultációt tarthat, sőt ugyanaz a diák ugyanannak a tanárnak más-más időpontokban tartott konzultációin is részt vehet. Ezekből következik, hogy sem a TANÁR, sem a DIÁK, sem pedig ez a két azonosító együtt nem kulcsa a relációnak. De egy személy egy időben csak egy helyen tartózkodhat. Ebből következik, hogy a TANÁR, IDŐPONT attribútumok kulcsot alkotnak, de ugyanilyen okból kifolyólag a DIÁK, IDŐPONT attribútumok is kulcsot alkotnak.

Vegyük észre azt, hogy a kulcsok nem önkényes döntések következtében alakulnak ki, hanem az adatok természetéből következnek, mint a funkcionális vagy a többértékű függőség.

A relációban idegen/külső kulcsot vagy kulcsokat is megkülönböztetünk. Ezek az attribútumok nem az adott relációban, hanem az adatbázis másik relációjában alkotnak kulcsot. Például ha a KONZULTÁCIÓ relációban a DIÁK azonosítására a személyi számot alkalmazzuk, akkor ez egy külső kulcs a személyi adatokat nyilvántartó relációhoz.

Adatmodell hibák

Anomáliák:

Hibák, melyek nem megfelelően kialakított adatmodell miatt az adatbázis inkonzisztenciájához vezethetnek (mert nem csak egy egyedre jellemző tulajdonságokat tárolunk egy táblában, vagy bizonyos tulajdonságokat többszörösen tárolunk).

Fajtái

- bővítési anomália: valamely táblába új rekord felvitele azért nem történhet meg, mert a táblában olyan attribútum értékek is szerepelnek, melyek felvitelekor vagy később sem állnak rendelkezésünkre.
- módosítási anomália: valamely attribútumot több táblában is tárolunk, de az attribútum-értékek módosításakor a módosítást nem mindenütt vagy nem egyformán végezzük el.
- törlési anomália: valamely táblában törlünk egy adatot, s elvesznek olyan fontos információk is, melyekre a későbbiekben még szükségünk lesz.

Redundancia:

átfedést jelent. Gyakorlatban legtöbbször a fizikai átfedést értjük alatta - többszörös adattárolást az adatbázisban -, tervezésnél leginkább ilyen fontos, hogy logikai átfedésekre is figyeljünk!

Fajtái

- Logikai átfedés:

Nyílt logikai átfedés: ugyanaz a tulajdonságtípus azonos elnevezéssel több egyednél is szerepel. Többszörös tárolást eredményez. Szükséges lehet biztonsági, vagy hatékonysági okból, vagy pl.: kapcsolatok megvalósításához (idegen kulcsként) Itt a logikai átfedés hiánya szintén hibának számít.

Rejtett logikai átfedés (szinonima jelenség): ugyanazt a tulajdonságot jelöljük különböző elnevezéssel.

Látszólagos logikai átfedés (homonima jelenség): különböző tulajdonságokra használjuk ugyanazon elnevezést.

- Fizikai átfedés: ugyanazon tulajdonságoknak vagy - szinonim névvel - egyednek többszörös tárolása az adatbázisban.

. Nézzük meg a következő relációt.

Tanár	**Tantárgy**	**Össz_óraszám**	**Tanított_órák**
Kiss Péter	Adatbázis kezelés	64	12
Nagy Andrea	Matematika	32	8
Szabó Miklós	Adatbázis kezelés	64	4

Kovács Rita	Matematika	32	5
	Angol	48	

Redundanciát tartalmazó reláció

A fenti relációban a tantárgyak össz óraszámát annyiszor tároljuk, ahány tanár tanítja az adott tantárgyat. A példa kedvéért feltételeztük, hogy egy tantárgyat több tanár is tanít. A redundancia a következő hátrányokkal jár:

- Ha egy tantárgy össz óraszáma megváltozik több helyen kell módosítani a relációban.
- Valahányszor egy új tanár kerül be a relációba ugyanannak a tantárgynak az előző soraiból kell elővenni az össz óraszám adatot.
- Az utolsó sorban szereplő tantárgy (angol) esetén még nem került kitöltésre a tanár személye. Új tanárnak a listára történő felvételekor ezt az esetet másként kell kezelni. Ilyenkor csak két üres értéket (tanár, tanított órák) kell átírni.

A redundancia fordul elő akkor is, ha levezett vagy levezethető mennyiségeket tárolunk a relációkban.

Levezetett adatokat tartalmazhat egyetlen reláció is abban az esetben, ha egyes attribútumok értéke egyértelműen meghatározható a többi attribútum alapján, például, ha a kerületet is nyilvántartjuk az irányítószám mellett. A redundáns adatok megszüntetésére két mód van. A levezetett adatokat tartalmazó relációkat vagy attribútumokat el kell hagyni. A relációkban tárolt redundáns tényeket a táblázatok szétbontásával, de kompozíciójával szüntethetjük meg (a 3.10 példában szereplõ relációt kettõ relációra bontjuk fel

Órák = {Tanár, Tantárgy, Tanított_Órák} és Össz_órák = {Tantárgy, Össz_óraszám}

A redundancia megszüntetése

A logikai tervezés célja egy redundancia mentes reláció rendszer, relációs adatbázis. A reláció elmélet módszereket tartalmaz a redundancia megszüntetésére, az úgynevezett normál formák segítségével. A következőkben a relációk normál formáinak definícióját mutatjuk be példákon keresztül. A normál formák előállítása során a funkcionális és a többértékű függőség, valamint a reláció kulcs fogalmát használjuk fel. A normál formák képzése során leegyszerűsítve, olyan relációk felírása a cél, melyekben csak a reláció kulcsra vonatkozó tényeket tárolunk. Öt normál formát különböztetünk meg. A különböző normál formák egymásra épülnek, a második normál formában levő reláció első normál formában is van. A

tervezés során a legmagasabb normál forma elérése a cél. Az első három normál forma a funkcionális függőségekben található redundanciák, míg a negyedik és ötödik a többértékű függőségekből adódó redundanciák megszüntetésére koncentrál.
A normál formákkal kapcsolatban két újabb a relációkhoz kapcsolódó fogalommal kell megismerkedni. Elsődleges attribútumnak nevezzük azokat az attribútumokat, melyek legalább egy reláció kulcsban szerepelnek. A többi attribútumot nem elsődlegesnek nevezzük.

Normálformák:

Első normál forma: a relációban minden érték elemi, a reláció nem tartalmaz adatcsoportot. A reláció minden sorában oszloponként egy és csak egy érték áll, az értékek sorrendje minden sorban azonos, minden sor különböző. Van legalább egy vagy több tulajdonság, amelyekkel a sorok egyértelműen megkülönböztethetők egymástól.
Mintaképpen álljon itt egy olyan reláció, melynek attribútumai is relációk.

Szakkör	**Tanár**	**Diákok**	
Számítástechnika	Nagy Pál	**Név**	**Osztály**
		Kiss Rita	III.b
		Álmos Éva	II.c
Video	Gál János	**Név**	**Osztály**
		Réz Ede	I.a
		Vas Ferenc	II.b

Szakkör	**Tanár**	**Diák**	**Osztály**
Számítástechnika	Nagy Pál	Kiss Rita	III.b
Számítástechnika	Nagy Pál	Álmos Éva	II.c
Video	Gál János	Réz Ede	I.a
Video	Gál János	Vas Ferenc	II.b

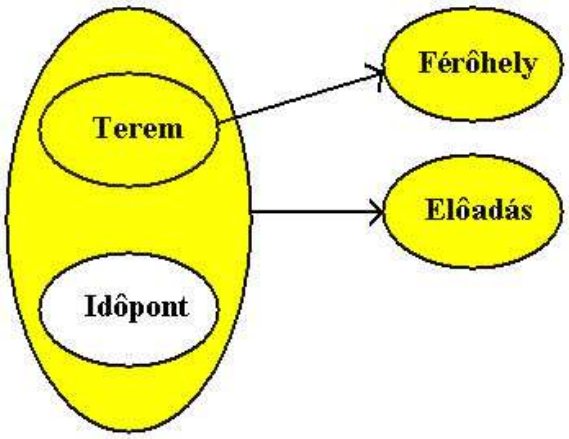
Második normál forma: a reláció első normálformában van, valamint egyetlen másodlagos attribútuma sem függ egyetlen kulcsának valódi részhalmazától sem. (Elsődleges

attribútumok, azok az attribútumok, melyek valamely kulcshoz tartoznak, másodlagos attribútumok, amelyekre ez nem teljesül.)

Konferencia			
Terem	**Időpont**	**Előadás**	**Férőhely**
B	10:00	Mitológia	250
A	8:30	Irodalom	130
B	11:30	Színház	250
A	11:00	Festészet	130
A	13:15	Régészet	130

Konferencia		
Terem	**Időpont**	**Előadás**
B	10:00	Mitológia
A	8:30	Irodalom
B	11:30	Színház
A	11:00	Festészet
A	13:15	Régészet

Konferencia		
Terem	**Időpont**	**Előadás**
B	10:00	Mitológia
A	8:30	Irodalom
B	11:30	Színház
A	11:00	Festészet
A	13:15	Régészet



Függőségi diagram

Nézzünk egy másik példát is a második normál forma feltételeit megsértő relációra. Egy épület energia gazdálkodásának ellenőrzésére az egyes helységekben rendszeresen megmérik a hőmérsékletet. A mérési eredmények értékeléséhez nyilvántartjuk az egyes helységekben található radiátorok számát is.

Hőmérsékletek

Terem	Időpont	Hőmérséklet	Radiátor
213	98.11.18	23	2
213	98.11.24	22	2
213	98.12.05	21	2
214	98.12.05	21	3
214	98.12.15	20	3

Konferencia

Terem	Időpont	Hőmérséklet
213	98.11.18	23
213	98.11.24	22
213	98.12.05	21
214	98.12.05	21
214	98.12.15	20

Termek	
Terem	**Radiátor**
213	2
214	3

Harmadik normál forma: a reláció második normálformában van és a másodlagos attribútumok között nincs funkcionális függés. Ha „B” attribútum értéke függ „A” attribútum értékétől, valamint „C” attribútum étéke tranzitíven függ „A” értékétől. A harmadik normál forma elengedhetetlen követelménye az ilyen tranzitív függések kiküszöbölése. Ha az adatbázis táblája nem harmadik normálformájú, akkor két táblára kell bontani úgy, hogy az egyes táblák külön-külön már harmadik normálformájúak legyenek.

Ezt ismét egy példa segítségével mutatjuk be.

Szakkörök		
Szakkör	**Tanár**	**Születési év**
Képzőművész	Sár Izodor	1943
Iparművész	Sár Izodor	1943
Karate	Erős János	1972

Szakkörök	
Szakkör	**Tanár**
Képzőművész	Sár Izodor
Iparművész	Sár Izodor
Karate	Erős János

Tanárok	
Tanár	**Születési év**

Erős János	1972
Sár Izodor	1943

Függőségi diagram

Boyce/Codd normál forma (BCNF)

A normál formák tárgyalása során eddig olyan relációkra mutattunk példákat, melyeknek csak egy reláció kulcsa van. A normál formák definíciója természetesen alkalmazható a több kulccsal rendelkező relációkra is. Ebben az esetben minden attribútum, mely valamely kulcsnak a része, elsődleges attribútum, de ez az attribútum függhet egy másik, ezt nem tartalmazó kulcs részétől. Ha ez a helyzet fennáll, redundanciát tartalmaz a reláció. Ennek a felismerése vezetett a harmadik normál forma egy szigorúbb definíciójához, a Boyce/Codd normál formához.

• Minden elsődleges attribútum teljes funkcionális függőségben van azokkal a kulcsokkal, melyeknek nem része

Mintaként tekintsük a következő relációt:

Tantárgyak

Tanár	Időpont	Tantárgy	Félév	Diák_szám
Kiss Pál	93/1	Adatbázis	1	17
Jó Péter	93/1	Unix	1	21
Kiss Pál	93/2	Adatbázis	2	32
Jó Péter	93/1	Unix	2	19
Kiss Pál	93/1	Adatbázis	3	25

Reláció harmadik normál formája és Boyce/Codd normál formát kielégítő dekompozíciója

Tételezzük fel, hogy minden tanár csak egy tantárgyat, de annak különböző féléveit oktatja. Ezek alapján a következő funkcionális függőségek írhatók fel:Tanár, Félév Tantárgy Tantárgy, Félév Tanár A relációnak két kulcsa van, a (Tanár, Időpont, Félév) és a (Tantárgy, Időpont, Félév). A relációban csak egy nem elsődleges attribútum található, a Diák_szám. Ez teljes funkcionális függőségben van mindkét reláció kulccsal, az elsődleges attribútumok között nincs függőségi viszony. Ezek alapján a reláció harmadik normál formában van. Azonban tartalmaz redundanciát, mivel ugyanazon tanár mellett többször is tároljuk a tantárgyat azonos időpontokban. A redundanciának az az oka, hogy a tanár attribútum az õt nem tartalmazó reláció kulcs (Tantárgy, Időpont, Félév) csak egy részétől (Tantárgy, Félév) függ.

Negyedik normál forma (4NF)

Sajnos még a Boyce/Codd normál forma is tartalmazhat redundanciát. Mindeddig csak a funkcionális függőségeket vizsgáltuk, a többértékű függőségeket nem. A további két normál forma a többértékű függőségekből adódó redundancia kiszűrését szolgálja.
Egy reláció negyedik normál formában van, ha egy XY többértékű függőséget tartalmazó relációban csak az X és Y-ban megtalálható attribútumokat tartalmazza.

Ötödik normál forma (5NF)

Hosszú ideig a negyedik normál formát tartották a normalizálás utolsó lépésének. A többértékű függőségek külön relációkban tárolásával azonban információt veszthetünk. Ennek bemutatására nézzünk egy példát. Egy számítógépes ismeretek oktatására szakosodott Kft több jól képzett tanárral rendelkezik. A tanárok többfajta tanfolyam oktatására is alkalmasak. A tanfolyamok az ország különböző részeiben kerülnek megtartásra.
Ezek alapján állítsuk össze a Tanár-Tanfolyam-Helyszin relációt.
Tanár-Tanfolyam-HelyszinTanárTanfolyamHelyszinNagy ÉvaAdatbázis I.SzegedKiss PálAdatbázis I.GyõrNagy ÉvaAdatbázis II.PécsKiss PálAdatbázis I.PécsTanár_TanfolyamTanfolyam-HelyszinTanár-HelyszinTanárTanfolyamTanfolyamHelyszinTanárHelyszinNagy ÉvaAdatbázis I.Adatbázis I.SzegedNagy ÉvaSzegedKiss PálAdatbázis I.Adatbázis I.GyõrKiss PálGyõrNagy ÉvaAdatbázis II.Adatbázis II.PécsNagy ÉvaPécsAdatbázis I.PécsKiss PálPécs

3.20 ábra Reláció, dekompozíciója és ötödik normál formája

A következő függőségeket írhatjuk fel Tanár->>Tanfolyam Tanár->>Helyszín Tanfolyam->>Helyszín.

Az egyetlen reláció kulcs tartalmazza az összes attribútumot (Tanár, Tanfolyam, Helyszín), ebből következik, hogy Boyce/Codd normál formában van a reláció, de mégis tartalmaz redundanciát. Például két sorban is megtalálható, hogy Kiss Pál Adatbázis I. tanfolyamot tanít. A relációt felbontva két - csak egy többértékű függőséget tartalmazó - relációra, (Tanár, Tanfolyam) és (Tanár, Helyszín), a redundancia megszüntetésével információt is vesztünk. A felbontás után már nem tudjuk, hogy a tanár melyik tantárgyát oktatja az adott helyszínen. Például Nagy Éva adatbázis I. vagy adatbázis II. tanfolyamot tart-e Pécsett. Az eredeti relációt három relációra felbontva kapjuk meg az ötödik normál formát. Az eredményül kapott három reláció összekapcsolásával előállítható az eredeti reláció, de bármelyik két reláció összekapcsolása még nem elegendő.

A normál formák tárgyalása végén megjegyezzük, hogy a harmadik normál formáig mindenféleképpen érdemes normalizálni a relációkat. Ez a redundanciák nagy részét kiszűri. Azok az esetek, melyekben a negyedik illetve az ötödik normál formák alkalmazására van szükség, ritkábban fordulnak elő. Az ötödik normál forma esetén a redundancia megszüntetése nagyobb tároló terület felhasználásával lehetséges csak. Így általában az adatbázis tervezője döntheti el, hogy az ötödik normál formát és a nagyobb adatbázist vagy a redundanciát és a komplikáltabb frissítési, módosítási algoritmusokat választja.

Fizikai tervezés

A relációs adatbázisok esetében a logikai tervezés során a relációk már elnyerhetik végleges alakjukat, melyeket egyszerűen leképezhetünk az adatbázis kezelőben. A fizikai tervezés során inkább arra koncentrálunk, hogy a logikai szerkezet mennyire felel meg a hatékony végrehajtás feltételeinek, illetve milyen indexeket rendeljünk az egyes relációkhoz. A relációkon végrehajtott művelet együttest tranzakciónak nevezzük és általában a tranzakciók gyors végrehajtását kívánjuk elérni.

A fizikai tervezés során előfordulhat, hogy a relációkba szándékosan redundanciákat építünk a hatékonyabb tranzakció kezelés érdekében. Ez visszalépésnek tűnhet a logikai tervezés során követett következetes redundancia megszüntető manipulációkhoz képest. A lényeges különbség viszont az, hogy itt a redundancia ellenőrzött módon kerül be a relációba, nem csak

véletlenül maradt ott a hiányos tervezés miatt. Gyakran előfordul például az, hogy a sűrűn együtt szükséges adatokat egy relációban tároljuk a lehető leggyorsabb visszakeresés érdekében.

A tervezés támogatása szoftverekkel

A tervezési módszertan nem tér el az eddig bevett gyakorlatokról. Minden esetben követi a szoftverfejlesztési sztenderdek által lefektetett iránymutatókat, melyek az alábbiak,

1. Követelmények tervezése
2. Adatbázis és az adatbázis táblák és annak mezőinek tervezése
3. Az adatbázis táblák közötti táblák meghatározása
4. Stressz és normál tesztelések

Követelmények tervezése:

Jelen fejezet előtt már meghatároztuk a követelményeket. Fontos, hogy a követelmények határozzák meg azokat az elvárásokat, melyeknek meg kell, hogy feleljen az adatbázis. Amennyiben eltérés mutatkozik a követelmény és a terv között, addig nem szabad tovább haladni a folyamatban, amíg minden eltérés nem lett átgondolva és rögzítve.

Adatbázis és az adatbázis táblák és annak mezőinek tervezése:

Amikor egy követelményt meghatároztunk az ott található információkat és az adatbázis célját összevetve meghatározható, hogy milyen táblarendszer akarunk létrehozni. Így mindig kellő figyelmet kell fordítani arra, hogy az adatok, melyek a követelmények megvalósításához szükségesek olyan struktúrát alkossanak, melyek megfelelnek az általános adatbázis elmélet alapjainak. Amennyiben le tudtuk képezni a struktúrát állhatunk neki az adatbázis mezők létrehozásának. Ebben a fázisban jelöljük ki az elsődleges kulcsokat is. Jelen feladatban az elsődleges kulcs minden esetben a tábla technikai (uid) azonosítója, mely az adatsort azonosítja. Viszont a historikus adatkezelési modell bevezetésével az adatbázisban található egyedet az referencia azonosítóval (rid) azonosítjuk. Így érhető el az, hogy a rendszerben bár több rid található, mégis egyedi rekordokat képeznek. Az egyediséget a rid és a dtm_ValidTo mezők együttes értelmezése kell, hogy megadja.

Az adatbázis táblák közötti táblák meghatározása:

Amennyiben a technikai azonosítóhoz (uid)-hez fordulunk az egyed és a historikus adatkezelés elvész. Emiatt a kapcsolatokat minden esetben a rid-hez azaz a referencia azonosítóhoz kell rendelni. Amennyiben ezt nem így tennénk a historikus adatkezelés nem valósulhat meg.

Stressz és normál tesztelések:

A tesztelés kulcsfontosságú része a fejlesztésnek. Mivel az adatszerkezetben több rekord is szerepelhet egyszerre, emiatt fontos, hogy a tervezési szakaszt követően a fizikai megvalósítás közben állandóan teszteljük az adatbázis szerkezetet. Az adatbázis szerkezetet nem csak a normál adat felvitel, módosítás, törlés metódusokkal szükséges tesztelni, hanem stressz tesztelni is szükséges. Ez azt jelenti, hogy intenzív adatbevitelt kell megvalósítani, hogy a határokat be lehessen állítani.
Ilyen határértékek lehetnek az alábbiak,

- Milyen részletességgel lehet kiszámoltatni egy felületet?
- Adatok mennyisége és lekérdezések gyorsasága közötti kapcsolat
- Megfelelő adatok bevitele

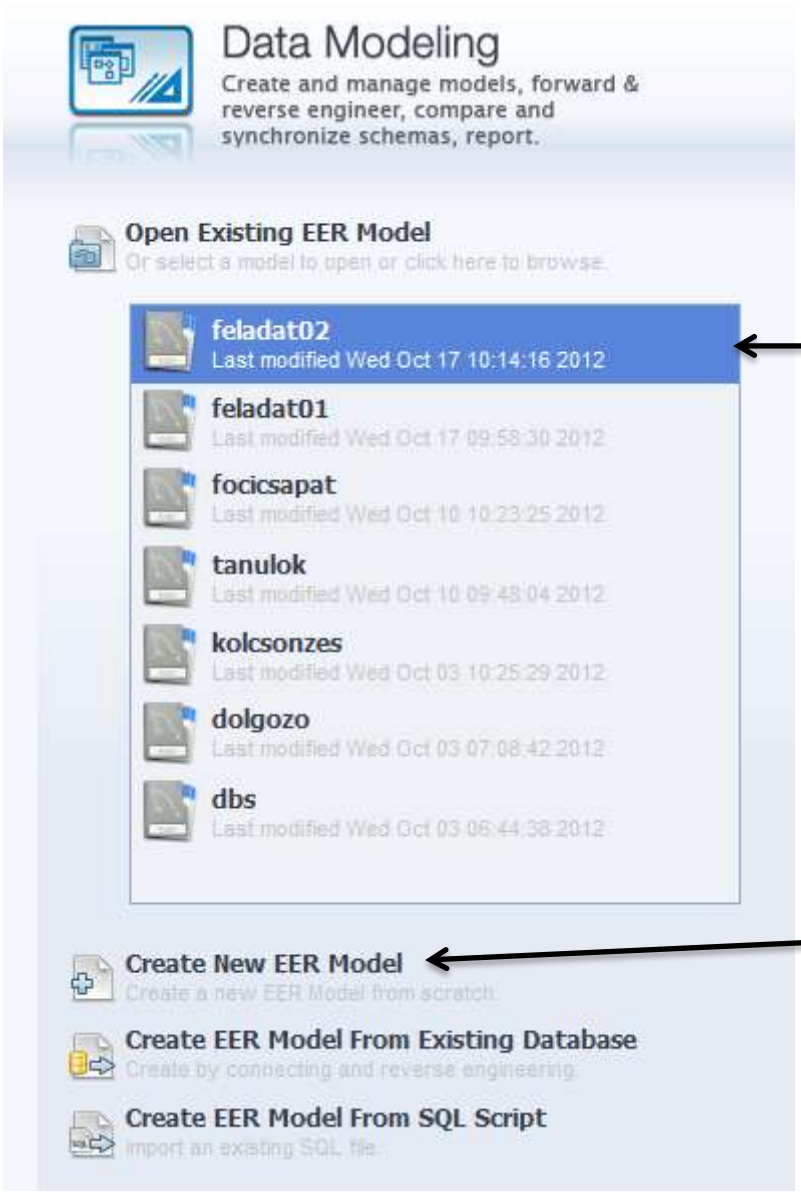
A stressz teszteléshez a JMeter alkalmazást használjuk, mely képes több szálon egyszerre futtatni lekérdezéseket, illetve adatok lekérdezését is egyszerre tudja megvalósítani.

A MySQL WorkBench

Első lépések: Ismerkedés a felülettel

Modell létrehozása (Data Modelling).

A modellben vannak a táblák, a kapcsolatok, kulcsok és stb.

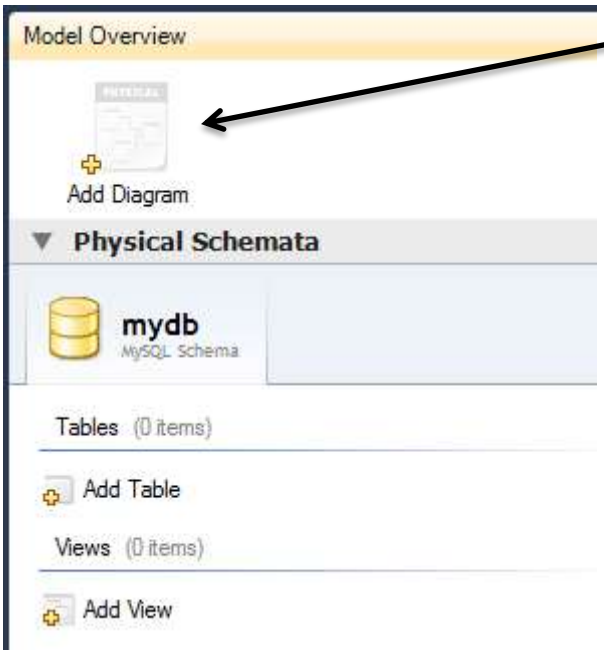


Itt található a már elkészített modellek, melyeket újra megnyithatunk és változtathatjuk.

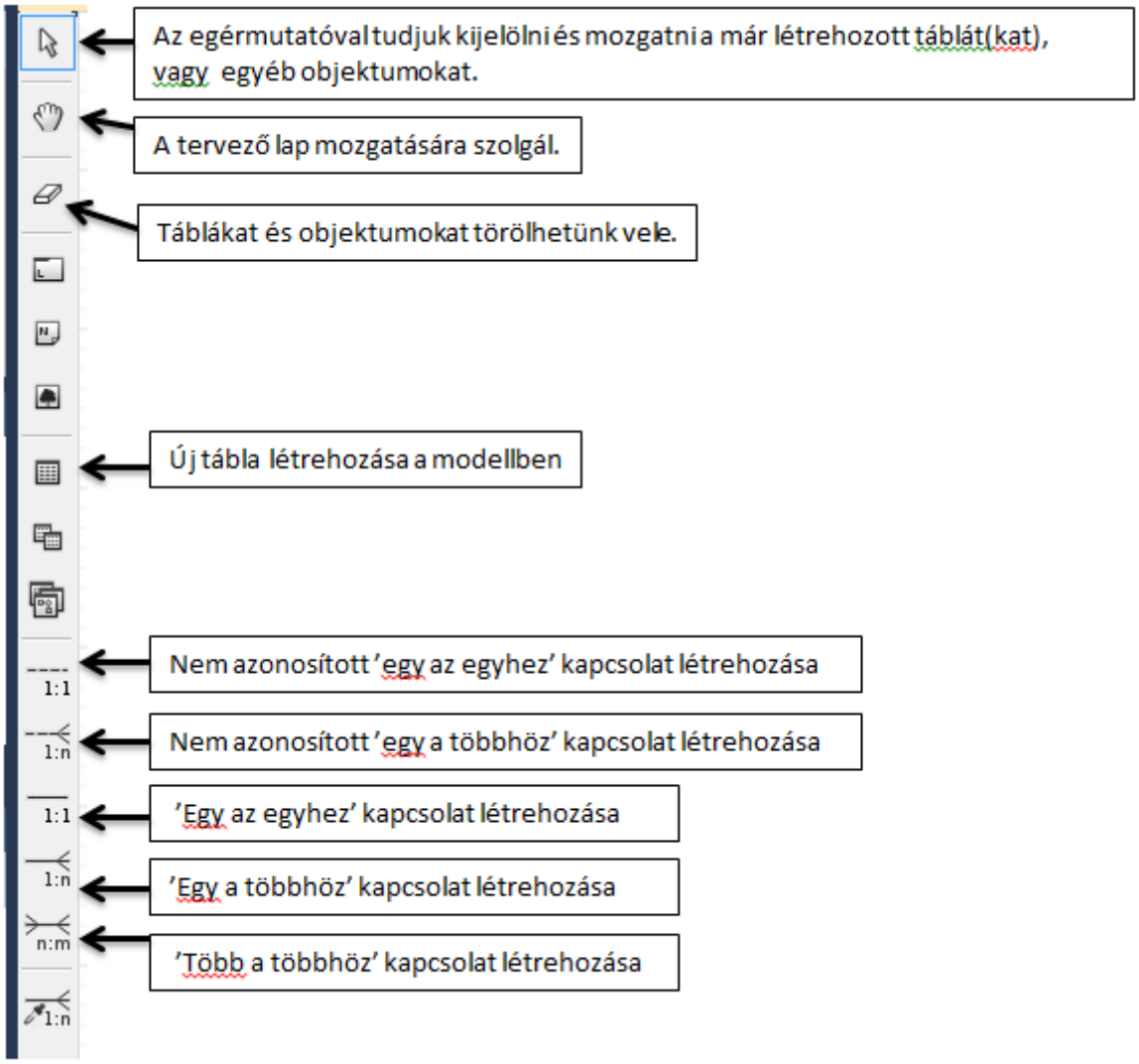
A modell nézetben könnyedén megtervezhetjük az adatbázis tábláit, a mezők típusait és a táblák közötti kapcsolatokat.

Erre kattintva
új modellt hozhatunk létre

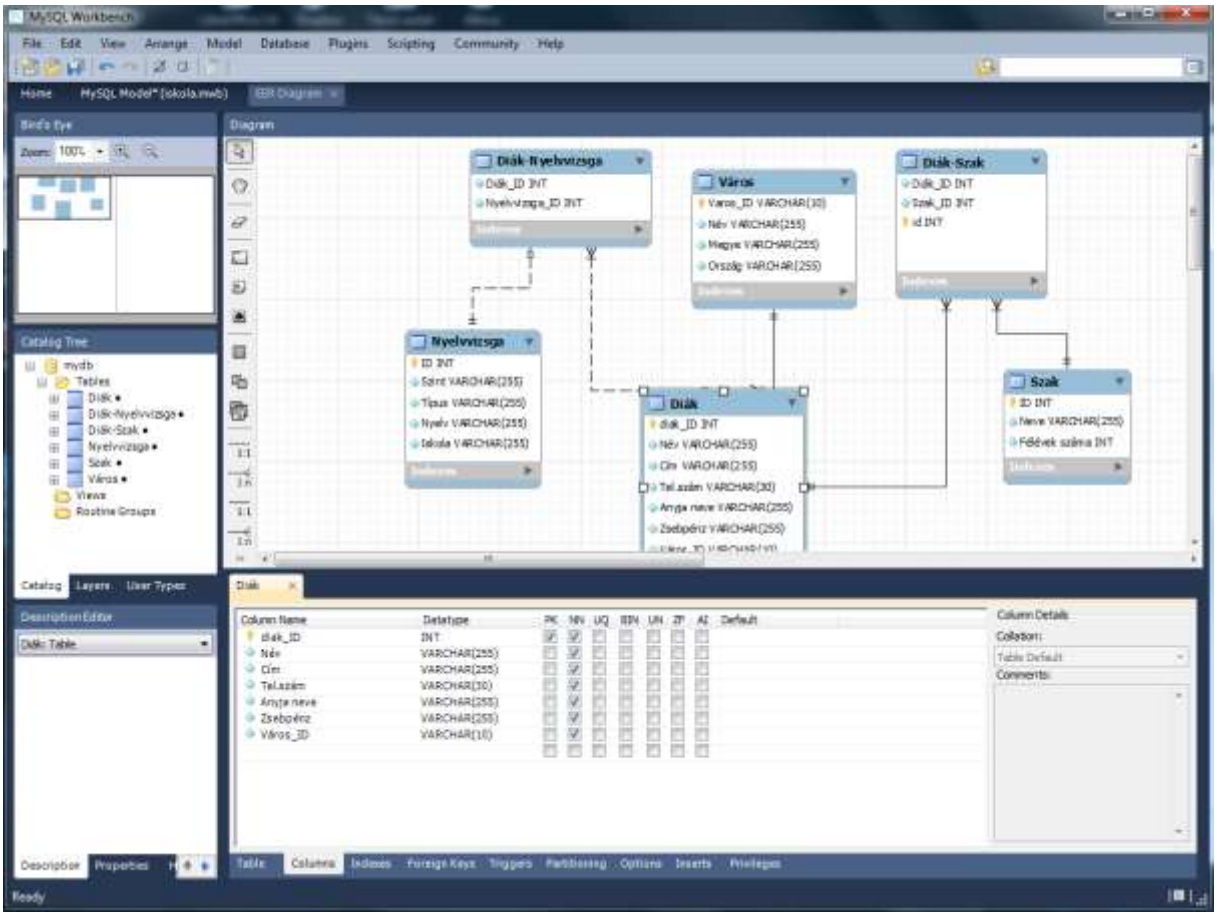
Kattintsuk a Create New EER Model-re, ekkor megjelenik egy ilyen ablak:



Ide kattintva egy új modellt hozhatunk létre.

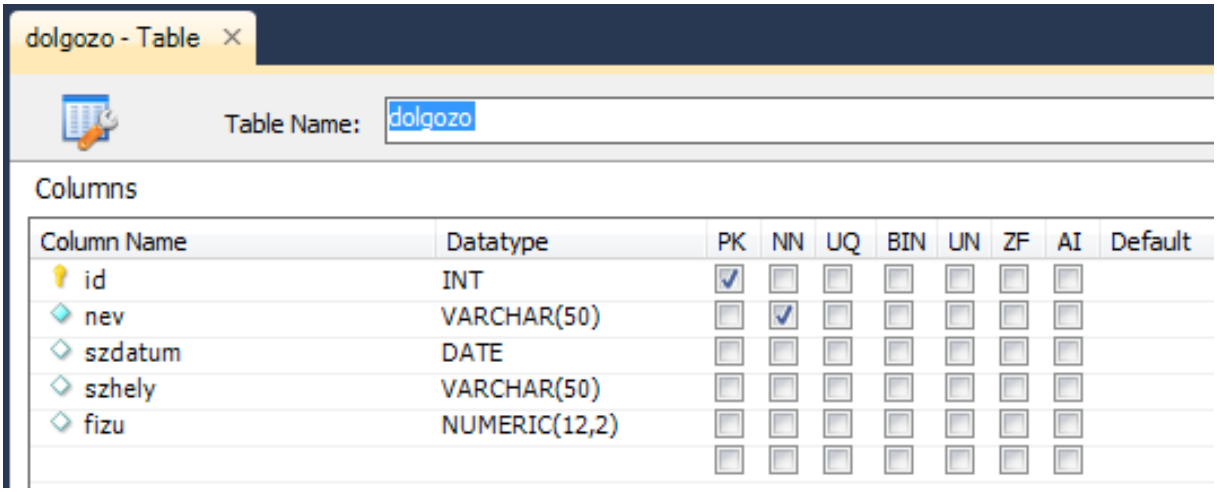


Mezők tervezése:



Egy tábla tulajdonságainak szerkesztése közben.

Megadhatjuk a mezők típusát és egyéb jellemzőit.



A ’dolgozo’ tábla elsődleges kulcsa az ’id’ mező, melynek típusa ’int’.

A mezők megnevezése után választjuk ki a mezők típusát.

A mezők az alábbi tulajdonságokkal is rendelkezhetnek:

PK – PrimaryKey – Elsődleges kulcs

NN – NotNull – A mező nem lehet üres

UQ

BIN

UN

ZF

AI

A megfelelő négyzetek kipipálásával adhatjuk meg a mezők további tulajdonságait.

Kapcsolatok létrehozása:

Kiválasztjuk a kapcsolat típusát és összekötjük a összekötni kívánt mezőket, ügyelve arra, hogy a reláció ’egy’ oldala mindig az elsődleges kulcshoz tartozzon, a ’több’ oldal pedig az idegen kulcshoz.

Arra is figyelni kell, hogy a kulcsos mezők típusa megegyezzen.

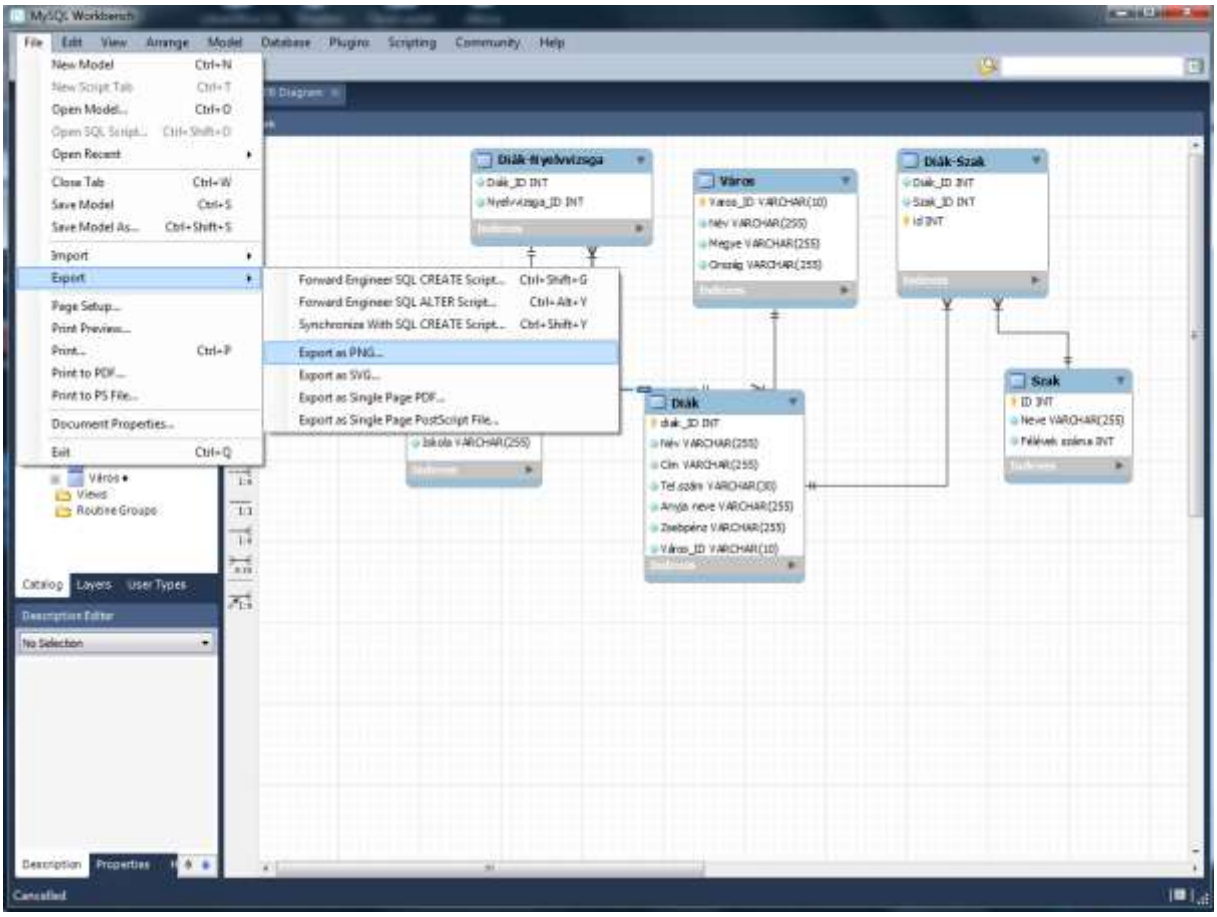
Több a többhöz kapcsolat létrehozása kapcsoló táblával történik. A tábla nevében megnevezzük, hogy egy kapcsolt tábla lesz, majd az összekapcsolt mezők neveit is jelezzük. Itt is igaz, hogy figyelni kell, hogy a kulcsos mezők típusa megegyezzen.

Kapcsolatok törlése:

Ez esetben a program rákérdez, hogy az összekapcsolt mezőket is törölje vagy változatlanul hagyja.

Ha az egeret a kapcsolatot szimbolizáló vonal fölé húzzuk, akkor láthatjuk, hogy mely mezők összekapcsolását jelölni a kiválasztott vonal.

A megtervezett és jól láthatóan elrendezett adatbázis tervet akár exportálhatjuk is valamilyen kép formátumba, hogy egy esetleges bemutatón könnyen felhasználható legyen.



Relációs algebra műveletei

A relációkkal kapcsolatban a matematika egy külön ága fejlődött ki. A matematikusok műveleteket definiáltak a relációkra és a halmaz műveleteket is alkalmazták a relációkra. Röviden ismertetjük ezeket a műveleteket.

Szelekció

A szelekció művelete során egy relációból csak egy adott feltételt kielégítő sorokat őrizzük meg az eredmény relációban. 1.1 ábra Szelekció, horizontális megszorítás

Projekció

A projekció során egy reláció oszlopai közül csak bizonyosakat őrzünk meg az eredmény relációban.

... 1.2 ábra Projekció, vertikális megszorítás

Descartes szorzat

A Descartes szorzat két reláció sorait minden kombinációban egymás mellé teszi az eredmény relációban.

... 1.3 ábra Descartes szorzat

Összekapcsolás

Az összekapcsolás művelete két vagy több relációt kapcsol össze egy-egy attribútum érték összehasonlításával. Az összekapcsolás leggyakoribb esete amikor az attribútumok egyezését vizsgáljuk, ezt egyen összekapcsolásnak nevezzük.

Ez egy speciális szorzás mely a következő műveletsorral írható le

1. Vegyük az első reláció egy sorát
2. Az összekapcsolási feltételt vizsgáljuk meg a második táblázat összes sorára, ha igaz, adjuk mindkét reláció sorát az eredményhez
3. Folytassuk az 1. ponttal amíg van még sor az első relációban

Az összekapcsolás eredmény relációjában az első reláció csak azon sorai szerepelnek, melyekre található a feltételt kielégítő sor a második relációban. Gyakran arra van szükség, hogy az első reláció valamennyi sora szerepeljen legalább egyszer az eredmény relációban. Ezt a fajta összekapcsolást külső összekapcsolásnak nevezzük.

Halmaz műveletek

A halmazokkal kapcsolatos alapvető műveleteket, unió metszet, különbség, a relációkra is értelmezzük. Minden értelmezett halmazművelethez legalább két operandus szükséges, a különbség esetében több sem lehet. A halmaz műveletek csak azonos szerkezetű relációk között hajthatók végre, ez alatt azt értjük, hogy a műveletbe bevont két reláció oszlopainak meg kell egyeznie az elnevezésben és a tárolt adat típusában is. A relációkra általában a komplemens képzés nem értelmezhető.

Unió

Az unió művelete azonos szerkezetű két vagy több reláció között végezhető el. Az eredmény reláció tartalmazza azokat a sorokat, melyek a műveletbe bevont relációk közül legalább egyben szerepelnek. Ha ugyanaz a sor az egyesítendő relációk közül többen is szerepelne, akkor is csak egyszer szerepel az eredmény relációban.

... 1.4 ábra Az unió művelet

Metszet

A metszet művelete azonos szerkezetű két vagy több reláció között végezhető el. Az eredmény reláció csak azokat a sorokat tartalmazza, melyek a műveletbe bevont relációk közül mindegyikben szerepelnek.

... 1.5 ábra A metszet művelet

Különbség

A különbség művelete azonos szerkezetű két reláció között végezhető el. Az eredmény reláció csak azokat a sorokat tartalmazza, melyek a első relációban megtalálhatóak, de a másodikban nem.

Feladatok

1. Alkotás – Alkotó
Az alkotásról tudjuk, hogy melyik évben készült, mi a műfaja (szobor, festmény, rajzkönyv stb) és mennyi az értéke forintban, ismerjük a címét, a művészt, a művész születési évét, a nemzetiségét. Egy alkotás mindig egy műfajhoz tartozik, és a művészek is egy nemzetnek fiai. A művészek akár teljes munkásságát is tudjuk feldolgozni!
Leadandó az adatbázis terv táblanevekkel, mezőnevekkel, mezőtípusokkal, kapcsolatokkal rajzban.

2. A Stupido-Gigantic GmbH Ltd Kft S.A. főleg kereskedelemmel foglalkozik. Ennek megfelelően *rendelések* futnak be hozzá *vásárlók*tól, amelyeket a *szállítók* révén teljesít. Egy terméket csak egy szállító szállít, és egy szállító csak egy terméket szállít. Nyilvántartjuk a vásárlók pillanatnyi egyenlegét. Egy rendelésen több *tétel* is szerepelhet, a tétel tartalma a termék megnevezése és a mennyiség. A szállítók, termékek és városok neve egyedi, de a vásárlók neve csak egy városon belül egyedi. Feltehetjük, hogy minden városnak csak egy irányítószáma van.
A tárolt adatok ömlesztve:
SzállítóIrányítóSzám, VásárlóIrányítóSzám, SzállítóNév, RendelésSzáma, Mennyiség, RendelésDátuma, VásárlóVárosnév, Egyenleg, Egységár, Terméknév, SzállítóVárosnév, Vásárlónév, Terméknév.
Készítsük el a Stupido-Gigantic GmbH Ltd Kft. S.A. kereskedelmének logikai adatbázis-modelljét (a 3NF alakig)!

3. Tervezzen egy adatbázist, melyben egy feladatmegoldó verseny eredményeinek nyilvántartását tudjuk tárolni. Ismerjük a versenyző nevét, iskolája nevét, az iskola részletes címét. A versenyző osztályát, amiből érkezett. Valamint a versenyen elért feladatonkénti pontokat, melyeket feladat sorszám, pontszám formában kapunk meg. Nincs megkötés, hogy egy iskolából hány tanuló jelentkezhet a versenyre.

4. Egy arab sejk szeretné nyilvántartani a kőolaj kereskedéshez kapcsolódó adatait. A sejk az alábbi adatokat szeretné látni a nyilvántartásában: Kőolaj kút: (Név, kapacitás), Kőolaj finomítók(Név, Cím, kapacitás), Vevők(Név, Cím). A sejk foglalkozik mind finomított mind nyersolaj eladásával. Éppen ezért szeretné azt is követni, hogy egyes vevői mit vásároltak. A nyersolaj értékesítését a kőolaj kutak végzik, míg a finomított olaj eladását a kőolaj finomítók végzik. Szeretné tudni a sejk, hogy az egyes vevők milyen határidővel, mekkora rendelést vettek fel A finomítók a sejk kőolaj kútjaiból származó olajt dolgozzák fel, és a sejk szeretné azt is nyomon követni, hogy az egyes kutak mely finomítókba szállítanak olajt, a szállítás határidejét, és a mennyiségét is szeretné nyilvántartásba venni.

5. Egy videokölcsönző nyilvántartását vizsgáljuk. A kölcsönző kizárólag videofilmeket kölcsönöz, a videofilmekről nyilvántartást vezetnek, mely nyilvántartás tartalmazza a film címét, főszereplőjét, rendezőjét. A filmeknek egyedi azonosító száma van, a cím nem biztos, hogy egyedi. Egy film több kazettán is szerepelhet, a kölcsönző által készített másolatokban. A kazettáknak szintén egyedi azonosítójuk van, valamint a típusát is nyilvántartják. A kölcsönzőből kizárólag tagok kölcsönözhetnek. A tag nevét, címét, valamint egyedi sorszámát nyilvántartják. A tagok filme(ke)t kölcsönöznek, a kikölcsönzés dátumát és a visszahozás dátumát is pontosan követni kell. A kölcsönző egy nagykereskedőtől vásárolja a filmeket, minden rendelésnek van egy száma és a visszakeresés érdekében a rendelés dátumát is nyilvántartják. Egy rendelésen több film is szerepelhet.
A videokölcsönzőnél található dokumentumok a következő adatokat tartalmazzák a különböző adatlapokon. Ezekből a dokumentumokból kell megalkotni a logikai adatbázis-modellt:

Kazetta száma
Kazetta típusa
Film címe
Film száma
Rendelés száma
Rendelés dátuma
Kölcsönzés száma
Tag sorszáma
Tag neve
Tag címe
Kölcsönzés dátuma
Visszahozás dátuma
Film főszereplője
Film rendezője

6. készítsünk egy adatbázist, mely tartalmazza egy iskola tanulóinak nevét, címét, otthoni telefonszámát, anyja nevét, havi zsebpénzét, mikor kezdte az iskolát, milyen szakos (testnevelés, ének-zene, angol, német), nyelvvizsgái (angol, német, francia, orosz, …). Figyeljünk oda a cím részletezésére! A város ismétlődésére. A legáltalánosabb esetet tervezzük meg. Beadandó a 3NF-ben lévő adatbázis terv.

7. Egy szálloda szeretné nyilvántartani az egyes vendégek kiadásait. A szállodában a vendégek szobákban szállhatnak meg egy adott időszakra. A szobáknak fontos tulajdonsága az alapterülete, ágyak száma és a napi díja. A vendégek több szobában is megszállhatnak, valamint egy szobában, akár egyszerre is, több vendég szállhat meg. A vendégek szobahasználatához kacsolódó fontos információ a bejelentkezés és a kijelentkezés időpontja. Természetesen lehet olyan szoba is, amelyben még nem szálltak meg vendégek. A vendégek foglalhatnak szobákat is előre, a foglaláshoz kötődő fontos információ az érkezés és elutazás tervezett dátuma. A vendégek a szállodában különböző szolgáltatásokat vehetnek igénybe (internet, étkezés, szauna …), a szolgáltatásokhoz kapcsolódó információk a szolgáltatás neve, ára, leírása. A vendégek több szolgáltatást is igénybe vehetnek, valamint lehetnek olyan szolgáltatások is, melyet még senki sem vett igénybe.

8. Tervezzen meg egy úszó eredményeket nyilvántartó adatbázist. A versenyzők különböző versenyeken elért időeredményeit. Az adatbázis tartalmazza:

- az egyesületek adatait
- az úszók adatait
- a versenyek adatait
- a verseny helyszínek adatait
- az elért eredményeket

Még lehet tervezési feladat

Az SQL nyelv alapjai

A DDL elemei

Sémák létrehozása, a create

A táblákat a CREATE TABLE paranccsal lehet létrehozni. Egy táblában <=1000 oszlop lehet. Több adattípust is lehet alkalmazni, de gyakorlatilag csak néhányra van szükség.

NUMBER(*h, t*) – numerikus adatok, hossza<=38, *h* a szám hosszát tartalmazza, *t* – a tizedes-jegyek számát;

DATE – dátum és idő típus;

VARCHAR2*(méret)* – karakter típus, a hossza váltózó (*max* hossza<=4000);

CHAR(*méret*) – karakter típus, fix hosszúságú (*max* hossza<=2000);

NCHAR(*méret*) – azonos a CHAR típussal, de a *max* méret függ a karakterkészletétől,

LOB (*large objects*) – bináris v. szöveges formátum. Lehet kép-, hang-, vagy nagy méretű szöveges állomány.

A LOB típuson belül lehetnek:

LONG – szöveges adatok, max hossza<=2 Gbyte;

LONG RAW – bináris adatok, max hossza<=2 Gbyte;

A LONG és LONG RAW típusokat csak egyszerű adatfeldolgozásban alkalmazzák.

CLOB, NCLOB, BLOB – *belső LOB típusok*, mivel az AB-ban tárolódnak, *max* hossza<=4 Gbyte;

CLOB – az adatokat tárolja az AB karakter-készlete alapján;

NCLOB – az adatokat tárolja a nemzeti karakter-készletben.

Az Oracle mindegyik táblába automatikusan helyez egy ROWID nevű pszeudooszlopot, amely a programozók ritkán szokták használni.

ROWID – pszeudooszlop, amely tartalmazza a sor logikai egyedi címét. A ROWID nem változható meg, de a SELECT parancsban lekérdezhető:

```
SELECT rowid FROM tabla_1;
```

A *Sor sorszáma* – olyan szám, amely akkor rendelődik a sorhoz, amikor az bekerül a táblába. Ez a szám része a ROWID-nek.

Csak az a felhasználó hozhat létre táblákat, aki CREATE TABLE vagy CREATE ANY TABLE *privilégiummal* rendelkezik.

A STORAGE paraméter alkalmazása a CREATE TABLE parancsban

Egy táblát a **STORAGE** memória-paraméterek nélkül is lehet létrehozni. Ebben az esetben a tábla az alapértelmezési értékeket kapja, ami nem mindig jó megoldás. A memória egységeket (**szegmenseket)**, melyekben a táblák tárolódnak, ***extentnek*** nevezzük. Az első extent neve **INITIAL**, a többi extentek pedig *másodlagos extentek*.

```
CREATE TABLE testtab
 (col1 VARCHAR2(10))
 STORAGE (INITIAL 100K
       NEXT 50K
       MINEXTENTS 1
       MAXEXTENTS 99
       PCINCREASE 20);
```

Az első extent 100K memóriát kap, a második – 50K. A **PCINCREASE 20** érték alapján mindegyik következő extent mérete 20% növekszik az előzőhez képest. **MINEXTENTS** 1 azt jelenti, hogy a tábla először 1extentent kap, a **MAXEXTENTS** 99 – a tábla maximálisan 99 extentet kaphat.

A táblákat, mint az indextáblákat is, particionálhatjuk. A particionálás lehetőséget ad a nagyon nagy táblák és indextáblák kezelésére. A particionálás során a nagy táblát kisebb és könnyebben kezelhető részekre osztja a rendszer.

Partíciónált tábla – olyan tábla, amelynek sorai kisebb azonos szerkezetű táblákba, *partíciókra,* vannak szétosztva. A partíciók fizikailag különböző helyeken tárolhatjuk.

A partíciónálás előnyei:

- Az I/O-terhelés jobb eloszlása
- Biztonsági mentés és visszaállítás egyszerűbb végrehajtása
- Az adatsérülés lehetőségének minimalizálása
- Az archiválási folyamat segítése.

Példa.

```
CREATE TABLE nagy_tabla
 (col1 VARCHAR2(10),
 (col2 VARCHAR2(10))
            )
```

```
 PARTITION BY RANGE (col1, col2)
       (PARTITION p1 VALUES LESS THEN (…)  TABLESPACE p1,
        PARTITION p2 VALUES LESS THEN (…)  TABLESPACE p2);
```

Sémaelemek módosítása, az alter

Az oszlop hosszát lehet növelni még akkor is, ha a tábla adatokat tartalmaz. Az oszlop hosszát pedig csak akkor lehet csökkenteni, ha a tábla üres.

ALTER TABLE testtab ADD col2 VARCHAR2(100);
ALTER TABLE testtab MODIFY col2 VARCHAR2(150);
ALTER TABLE testtab STORAGE (NEXT 10K
MAXEXTENTS 50);
ALTER TABLE testtab DROP col1;

A táblákat átnevezhetjük:

RENAME testtab TO testtab1;

és törölhetjük:

DROP TABLE testtab;

Példa.

```
CREATE TABLE students(
    id                NUMBER(5) PRIMARY KEY,
    first_name        VARCHAR2(20),
    last_name         VARCHAR2(20),
    major             VARCHAR2(30),                -- szak
    current_credits   NUMBER(3) );         -- leadott vizsgák száma
```

Az ORACLE-rendszer tartalmazz egy bemutató sémát, amelynek *Scott* a tulajdonosa:

```
CREATE TABLE scott.dept
 (deptno  NUMBER(2) NOT NULL,
  dname  VARCHAR2(14),
  loc  VARCHAR2(13),
 CONSTRAINT pk_dept PRIMARY KEY (deptno));
```

```
CREATE TABLE scott.emp
 (empno  NUMBER(4) NOT NULL,
```

```
 ename  VARCHAR2(10),
 job    VARCHAR2(9),
 mgr    NUMBER(4),
 hiredate  DATE,
 sal       NUMBER(7,2),
 comm      NUMBER(7,2),
 sal       NUMBER(7,2),
 deptn     NUMBER(2),
CONSTRAINT pk_emp PRIMARY KEY (empno));
```

Amikor egy olyan táblára (vagy más Oracle-objektumra) hivatkozunk, melynek nem mi vagyunk a tulajdonosa, akkor a tábla nevén kívül meg kell adni a tulajdonos nevét is
tulajdonos.tábla_név

CONSTRAINT integritási megszorítás alkalmazása

Hivatkozási integritás biztosítja, hogy egy adott oszlop csak olyan értékeket vehessen fel, amelyek megjelennek egy másik oszlopban. Integritási megszorításként lehet előírni.
Megszorítás – egy szabály vagy korlátozás az adatok egy részén kerül ellenőrzésre.
Integritási megszorítás – olyan szabály, amely egy vagy több oszlop lehetséges értékeinek körét határolják be. A megszorítást a tábla létrehozásánál az oszlopokhoz lehet hozzárendelni.
A CONSTRAINT záradékot a CREATE TABLE vagy ALTER TABLE parancsokban használhatjuk. A legtöbb megszorítás a CONSTRAINT záradék nélkül is megadható.
Léteznek:
táblamegszorítások
oszlopmegszorítások
Táblamegszorítás – olyan integritási megszorítás, amelyet egy tábla több oszlopára alkalmazunk. *Oszlopmegszorítás* pedig egy olyan integritási megszorítás, amelyet egy tábla oszlopához rendelünk.

A *táblamegszorítások listája:*
UNIQUE
PRIMARY KEY
FOREIGN KEY
REFERENCES
ON DELETE CASCADE

CHECK

ENABLE VALIDATE

DISABLE

Az *oszlopmegszorítások:*

NULL

NOT NULL

A többi *oszlopmegszorítások* a *táblamegszorításokkal* azonosak.

A DISABLE opciót a integritási megszorítás vagy trigger kikapcsolására (nem törlésére) alkalmazhatjuk. Leggyakrabban az ALTER TABLE parancsban alkalmazzák.

```
ALTER TABLE tabla DISABLE UNIQUE oszlop, oszlop,..
ALTER TABLE tabla DISABLE PRIMARY KEY
ALTER TABLE tabla DISABLE CONSTRAINT megszorítás_név
DISABLE ALL TRIGGERS
```

A DML elemei

- **INSERT** – új sor beszúrása
- **UPDATE** – adatok módosítása a már létező sorokban
- **DELETE** – sorok törlése
- **TRUNCATE** – az összes sorok gyors törlése

Új adat felvitele, az insert parancs

Először az SQL*Plus-ban ellenőrizzük a *testtab* tábla struktúráját:

```
DESCRIBE testtab;
```

NAME	NULL ?	TYPE
COL1		VARCHAR2(10)
COL2		NUMBER(38)

```
INSERT INTO testtab(col1, col2) VALUES('szöveg', 123);
```

Ugyan ezt az eredményt kapjuk, ha az

```
        INSERT INTO testtab VALUES ('szöveg', 123);
```

parancsot alkalmazzuk.

A NULL-értékeket beszúrása-

```
        INSERT INTO testtab (col1, col2) VALUES ('szöveg', NULL);
```

vagy

```
        INSERT INTO testtab (col1) VALUES ('szöveg');
```

Arra is van lehetőség, hogy a táblába az INSERT parancs egyszerre több sort szúrjon be (e célból alkalmazzuk a beágyazott SELECT parancsot)

```
        INSERT INTO testtab (col1, col2) SELECT e_name, emp_no FROM
emp;
```

Tábla létrehozása egy másik tábla alapján

```
CREATE TABLE emp_copy AS
 SELECT * FROM emp;
```

```
CREATE TABLE emp_copy2 AS
 SELECT emp_no, e_name FROM emp
WHERE e_name LIKE '%a%';
```

Leírás

Adatok módosítása, az update

Az UPDATE parancs módosítja az összes olyan sorokat, amelyekre teljesül a WHERE-feltétel. Ha az UPDATE parancs WHERE- feltételt nem tartalmaz, akkor a módosítás minden sorban végrehajtódik. Ez nagyon ritkán fordul elő.

Példa.

```
UPDATE emp_copy2  SET e_name='Kovács' WHERE emp_no=7499;

UPDATE emp_copy2  ec  SET (emp_no, e_name) =
       (SELECT emp_no, e_name FROM emp e WHERE e.emp_no=ec.emp_no)
WHERE e_name LIKE 'I%';
```

Az utolsó parancsban *ec* és *e* a táblák másodlagos nevei. Az utolsó WHERE-feltétel az egész parancshoz tartozik (az UPDATE-hoz), és azokat a sorokat adja meg, amelyeket módosítani akarunk. A beágyazott SELECT parancs azokat az értékeket szerkeszti, melyek bekerülnek az *emp_copy2* táblába. A SELECT-ben a WHERE-feltétel összekapcsolja a két tábla megfelelő sorait.

Adatok törlése, a delete

```
DELETE FROM emp_copy2 WHERE emp_no=7876;
```

```
DELETE FROM emp_copy2 WHERE emp_no IN
(SELECT emp_no FROM emp_copy);
```

Mindegyik SQL parancsban függvényeket lehet alkalmazni.

Példa.

```
UPDATE emp_copy  SET e_name=UPPER( SUBSTR(e_name, 1, 1))        ||
LOWER(SUBSTR(e_name, 2, LENGTH(e_name)-1));
```

Ugyan ezt az eredményt kapjuk, ha az INITCAP függvényt alkalmazzuk.

Adatok gyors törlése

A tábla összes sorait gyorsan lehet törölni a TRUNCATE paranccsal:

```
TRUNCATE TABLE emp_copy;
```

A parancs végrehajtása után a tábla struktúrája megmarad, de az egyetlen sort sem fog tartalmazni.

Jogok és felhasználók kezelése, a DCL

Privilégium (jogosultság) a felhasználó számára lehetővé teszi, hogy az AB-ban bizonyos műveleteket hajthasson végre.

A privilégium kéttípusú lehet:

- rendszer-privilégium;
- objektum-privilégium.

A rendszer-privilégiumok lehetővé teszik adatdefiníciós és adatvezérlő parancsok végrehajtását, és az AB-ba való belépést. Az objektum-privilégiumok az AB objektumokkal való műveletekhez adnak jogosultságot. A felhasználó a létrehozása után egyetlen privilégiummal sem rendelkezik, és a privilégiumokat az AB adminisztrátortól később kapja. Amikor a felhasználó kap egy privilégiumot, utána az engedélyezett műveteket végrehajtatja a felsorolt AB-objektumaival.

Egy AB-hoz általános esetben több felhasználó is kapcsolódhat. Az Oracle-ban minden objektumnak van felhasználó-tulajdonosa. A *nem tulajdonos-felhasználó* csak akkor végezhet bizonyos műveleteket az objektumokkal, ha *megkapta* a megfelelő *privilégiumokat.* Létezik több mint nyolcvan *rendszer-privilégium:*

```
ALTER
DELETE
EXECUTE
INDEX
INSERT
REFERENCES
SELECT
UPDATE
```

…

A következő táblázat tartalmazza egyes privilégiumok és AB-objektumok közti lehetséges kapcsolatokat.

Privilégium	Table	View	Sequence	Procedure
ALTER	+		+	
DELETE	+	+		
EXECUTE				+
INDEX	+			
INSERT	+	+		
REFERENCES	+			
SELECT	+	+	+	
UPDATE	+	+		

A INSERT, UPDATE és REFERENCES privilégiumokat lehet a tábla egyes oszlopaihoz kötni. Ha megakarjuk tudni, hogy milyen rendszer privilégiumokat lehet alkalmazni, végre kell hajtani a következő parancsot:

```
SELECT UNIQUE privilege FROM dba_sys_priv;
```

Privilégiumok adományozása

Az AB adminisztrátor a privilégiumokat a GRANT paranccsal adhatja meg a felhasználóknak:

```
GRANT  privilégium_lista ON objektum_lista TO felhasználó_lista
[WITH GRANT OPTION];
```

```
GRANT  privilégium_lista ON objektum_lista TO szerepkör_lista
[WITH GRANT OPTION];
```

```
GRANT  privilégium_lista ON objektum_lista TO PUBLIC
[WITH GRANT OPTION];
```

A WITH GRANT OPTION esetén a felhasználó az adott privilégiumot más felhasználónak is tovább adhatja.

Példa.

```
 GRANT SELECT, UPDATE ON emp_copy TO test_user;
```

A *test_user* felhasználó az *emp_copy* táblával végrehajthatja a SELECT és UPDATE parancsokat. Ezekben a parancsokban nem elég a tábla nevét *emp_copy* megadni, mivel a *tábla tulajdonosát* is meg kell adni. Például, ha a *scott* felhasználó az *emp_copy* tábla tulajdonosa, akkor a helyes hivatkozás:

```
SELECT * FROM scott.emp_copy;
```

Bizonyos esetekben ez a követelmény problémát is okozhat. Például, ha a tábla tulajdonosa a program létrehozása és futtatási ideje között megváltozott, akkor ezt figyelembe kell venni az összes táblára való hivatkozásban. Ilyen esetekben célszerű alkalmazni a tábla *szinonimáját:*

```
CREATE SYNONIM test FOR  scott.emp_copy;
```

A *scott.emp_copy* tábla megkapta a *test* szinonimát. Ha ezek után megváltozik a tábla tulajdonosa, akkor továbbra is a táblára lehet hivatkozni a szinonima segítségével

```
SELECT * FROM test;
```

A PUBLIC opció esetén az adott privilégiumokat az összes felhasználóra érvényes, ami azt jelenti, hogy az objektum nyilvános. *A Nyilvános objektum elérhető, látható az összes felhasználó számára*.

```
GRANT SELECT, UPDATE ON emp_copy TO PUBLIC;
```

Ha a privilégium tartalmazza az ANY opciót, akkor az az AB összes táblájára érvényes. A következő privilégiumok esetén:

```
DELETE ANY TABLE
UPDATE ANY TABLE
INSERT ANY TABLE,
```

a felhasználó az AB összes tábláit módosíthatja, még akkor is, ha nem tulajdonosa a táblának. Ezt a legnagyobb lehetséges jogosultság, és csak az adminisztrátor szintű felhasználó kaphatja azt.

A SESSION_PRIVS nézetben megtalálhatók az aktuális privilégiumok. Az ALL_TAB_PRIVS és ALL_COL_PRIVS nézetekből pedig megtudhatjuk, hogy milyen privilégiumokkal rendelkeznek a felhasználók.

Szerepkörök (ROLE)

Szerepkör – a privilégiumok együttese, és lehetőség a felhasználók csoportosítására. A felhasználókat egy csoportba lehet beosztani, és a csoporton belül mindegyik felhasználó azonos jogosultsággal rendelkezik. Például, létrehozunk egy szerepkört

```
CREATE ROLE kozos;
```

utána a szerepkörhöz privilégiumokat kapcsolunk

```
GRANT INSERT ON table_a TO kozos;
GRANT INSERT ON table_b TO kozos;
GRANT INSERT, DELETE  ON table_c  TO kozos;
GRANT UPDATE ON table_d TO kozos;
GRANT DELETE ON  table_e TO kozos;
GRANT SELECT  ON table_f TO kozos;
```

Ha a *felh_1* és *felh_2* felhasználók megkapják a *kozos* szerepkört

```
GRANT kozos  TO x;
GRANT kozos  TO y;
```

akkor rendelkezni fognak az összes *kozos* szerepkörhöz tartozó jogosultságokkal.

Beépített szerepkörök

Az Oracle-ban léteznek beépített szerepkörök is:

```
CONNECT-
ALTER SESSION
CREATE CLUSTER
CREATE DATABASE LINK
CREATE SEQUENCE
CREATE SESSION
CREATE SYNONIM
CREATE TABLE
CREATE VIEW
RESOURSE-
CREATE CLUSTER
CREATE PROCEDURE
CREATE  SEQUENCE
CREATE TABLE
```

Jogosultság a programok végrehajtására

Az AB-ban tárolódhatók

Eljárások (PROCEDURE)

Csomagok (PACKAGE)

Függvények (FUNCTION).

Ezek az objektumok csak akkor érhetők el, ha a felhasználó EXECUTE privilégiummal rendelkezik.

Példa.

```
GRANT EXECUTE  ON my_package TO PUBLIC;
GRANT EXECUTE  ON my_func TO felh_2;
GRANT EXECUTE  ON my_proc TO felh_1;
```

Privilégiumok visszavonása

Az AB adminisztrátor a privilégiumot a felhasználótól a REVOKE parancs által visszavonhat:

```
REVOKE privilégium ON objektum FROM felhasználó
[CASCADE CONSTRAINTS];
```

A CASCADE CONSTRAINTS a REFERENCES privilégium megvonása esetén törli az összes hivatkozási integritási megszorítást, amelyet a felhasználó hozott létre (CASCADE – lépcsőzetes).

```
REVOKE UPDATE ON emp_copy FROM test_user;
```

Ez a parancs után a *test_user* felhasználó nem módosíthatja a *emp_copy* táblát, de a SELECT parancsot továbbra is alkalmazhatja.

```
REVOKE  SELECT ON classes FROM user_1;
REVOKE ALTER TABLE, EXECUTE ANY PROCEDURE FROM user_2;
REVOKE  kozos FROM felh_1;
```

A lekérdezések, a QL

A legalapvetőbb SQL parancs egy teljes tábla, illetve a tábla valamely oszlopainak lekérdezésére szolgál. Ez tulajdonképpen a projekció relációalgebrai művelet megvalósítása.

A select parancs alapjai

A parancs általános szintaxisa a következő:

```
SELECT [ALL|DISTINCT] * | <oszlopnévlista> FROM <táblanév>
```

A parancs első részében a * jel azt jelenti, hogy a lekérdezés a teljes táblára vonatkozik, minden oszlopot, mezőt látni szeretnénk a lekérdezésben.
Az <oszlopnévlista> segítségével adhatjuk meg azt, hogy a lekérdezés eredményeképpen melyik oszlopokat kívánjuk megjeleníteni, amennyiben nem az őszeset.
Lehetőség van különböző módosításokra, például a listában szereplő elemeket átnevezhetjük, esetleg kifejezéseket képezhetünk belőlük. Ennek módja, hogy a mezőneveket követően egy AS kulcsszó után megadjuk az alias nevet. Ez csak a lekérdezés idejére jön létre:

```
SELECT ne vas „NÉV” …
```

Emellett alkalmazhatunk különböző függvényeket is, amelyek egy része az aggregáló függvények, másik része pedig olyan függvények, amelyeket a gazdanyelv biztosít az SQL környezet számára.

Az eredménytáblában az <oszlopnév> paraméterben megadott oszlopnév szerepel.
A parancsban használható ALL és DISTINCT kulcsszavaknak speciális jelentése van. Előfordulhat, hogy egy lekérdezés végrehajtása után a keletkezett elemek között ugyanaz a rekord többször előfordul.
Ilyenkor az ALL opció alkalmazásakor, illetve alapértelmezés szerint az azonos előfordulások többszörösen fognak megjelenni az eredményben.
Amennyiben a DISTINCT opciót alkalmazzuk, akkor minden azonos előfordulás csak egyszer jelenik meg az eredménytáblában.

A számított mezők és az aggregáló függvények

Amennyiben az oszlopkifejezésben aggregáló függvényt használunk, akkor az eredménytáblában nem a tábla egyes előfordulásainak adatai jelennek meg, hanem azoknak a megfelelő aggregáltját fogja képezni a parancs. A tábla soraiból álló halmazra végrehajtódik a megfelelő függvény.
Az <oszlopkifejezés> a következő aggregáló függvényeket tartalmazhatja:

- **COUNT**: Megadja a tábla sorainak számát. A pontos eredmény elérése céljából használjuk a *-ot vagy az elsődleges kulcs mezőt paraméternek.
- **SUM**: Megadja a paraméterében szereplő oszlop adatainak az összegét az összes rekordra. Csak numerikus attribútumra alkalmazható.
- **AVG:** Megadja a paraméterében szereplő oszlop adatainak az átlagát az összes rekordra. Csak numerikus attribútumra alkalmazható.
- **MIN**: Megadja a paraméterében szereplő oszlop adatainak a minimumát az összes rekordra. Csak numerikus attribútumra alkalmazható.
- **MAX:** Megadja a paraméterében szereplő oszlop adatainak a maximumát az összes rekordra. Csak numerikus attribútumra alkalmazható.

Tekintsünk néhány példát az egyszerű lekérdezésekre:
Ehhez használjuk a dolgozó táblát, ahol egy cég dolgozóinak nevét, születési idejét, lakhelyének városát és havi fizetését tároljuk.

Dolgozo				
ID	**Neve**	**Szdatum**	**Varos**	**Fizetes**
12	Kiss Szilárd	1985.01.05	Eger	120000

13	Nagy József	1973.05.06	Eger	156000
15	Gipsz Jakab	1955.11.01	Miskolc	210000
17	Kovács Piroska	1996.07.15	Budapest	189000

1. példa

A teljes tábla lekérdezése a következő paranccsal történhet:

```
SELECT * FROM Dolgozo
```

2. példa:

Tegyük fel, hogy szeretnénk lekérdezni az egyes dolgozók nevét a fizetésükkel együtt. A megfelelő parancs a következő:

```
SELECT Neve, Fizetes FROM Dolgozo
```

A keletkezett eredménytábla a következőképpen néz ki:

Neve	**Fizetes**
Kiss Szilárd	120000
Nagy József	156000
Gipsz Jakab	210000
Kovács Piroska	189000

3. példa

Készítsünk olyan eredménytáblát, amely tartalmazza a dolgozók nevét, fizetését, valamint a 20%-kal megemelt fizetéseket. Az új oszlop neve legye A dolgozó emelt fizetése. A feladatot az alábbi paranccsal oldhatjuk meg:

```
SELECT Neve, Fizetes, 1.2*Fizetes AS „Emelt fizetés” FROM Dolgozo
```

A keletkezett eredménytábla a következő:

Neve	**Fizetes**	**Emelt fizetés**
Kiss Szilárd	120000	144000
Nagy József	156000	187200
Gipsz Jakab	210000	252000
Kovács Piroska	189000	226800

4. példa

Jelenítsük meg a dolgozók városát úgy, hogy a listában ne legyen két azonos érték.

A feladatot az alábbi paranccsal oldhatjuk meg:

```
SELECT DISTINCT Varos FROM Dolgozo
```

A keletkezett eredménytábla a következő:

Varos
Eger
Miskolc
Budapest

5. példa

Készítsünk olyan táblázatot, amely megadja a vállalat dolgozóinak számát, az összes, az átlagos, a legkisebb és a legnagyobb fizetéseket.

A feladatot a következő módon oldhatjuk meg:

```
SELECT COUNT(id) as „A dolgozók száma”,
 SUM(fizetes) as „Összes fizetés”,
 AVG(fizetes) as „Átlagos fizetés”,
 MIN(fizetes) as „Legkisebb fizetés”,
 MAX(fizetes) as „Legnagyobb fizetés”
FROM Dolgozo
```

A keletkezett eredménytábla a következő:

A dolgozók száma	Összes fizetés	Átlagos fizetés	Legkisebb fizetés	Legnagyobb fizetés
4	675000	168750	120000	210000

6. példa

Készítsünk olyan táblázatot, amely megadja a vállalat dolgozóinak nevét és születési évét. A születési évet tartalmazó oszlop neve legyen ’Születési év’.

A feladatot az Oracle függvényével oldjuk meg.

```
SELECT Neve, TO_CHAR(szdatum, ’yyyy’) as „Születési év” FROM dolgozo
```

Ahol a TO_CHAR függvény a szdatum mezőben tárolt dátum típusú értéket a megadott formátumúra alakítja, esetünkben 4 jegyű évvé.

A keletkezett eredménytábla a következő:

Neve	Születési év
Kiss Szilárd	1985
Nagy József	1973
Gipsz Jakab	1955
Kovács Piroska	1996

Szűrések, a where záradék

A kiválasztást végrehajtó parancsnál a FROM után a következő szintaxisnak megfelelően adhatjuk meg az alparancsot:

```
[WHERE <feltétel>]
```

A feltételben operanduszok és operátorok szerepelhetnek.
Az operátorok összehasonlító (<,>,<=,>=,<>), aritmetikai (+,-,*,/) és logikai műveletek (AND, OR, NOT), míg az operanduszok lehetnek konstansok, reláció attribútumok, azaz oszlopnevek, illetve függvényhivatkozások.
A műveletekre érvényesek a szokásos precedencia szabályok, amelyeket természetesen zárójelezéssel felülbírálhatunk. Ügyelnünk kell arra, hogy a kifejezésnek mindig logikai értéket kell szolgáltatni, ugyanis az eredménytáblába azok az előfordulások fognak bekerülni, amelyekre a megadott kifejezés igaz (TRUE) értéket szolgáltat.

Létezik néhány jól használható predikátum függvény
Ezek közül az első a **BETWEEN** predikátum függvény, amely a következőképpen használható:

```
[<oszlopkifejezés> BETWEEN <alsóérték> AND <felsőérték>]
```

Az **<alsóérték>**, illetve **<felsőérték>** valamely ismert elemi típusnak (numerikus, dátum) megfelelő konstans, az **<oszlopkifejezés>** ugyanilyen típusú kifejezés, amely oszlopnevekből van képezve. Eredményként azok a rekordok fognak eleget tenni a feltételnek, amelyekre a kifejezés értéke a két konstans közé esik. Esetleg egyenlő valamelyikkel. Tehát a BETWEEN zárt intervallumot ír le.
Ezért a feltételnek akkor van értelme, ha az <alsóérték> paraméterben megadott konstans kisebb, mint a <felsőérték> paraméterben megadott.
Természetesen a predikátum helyettesíthető összehasonlító operátorokból összeállított logikai kifejezéssel:

```
fizetes between 100000 and 200000
```

ugyan azt jelenti, mint

```
(fizetes >= 100000) and (fizetes <= 200000)
```

A következő az **IN predikátum**, amely a következőképpen használható:

```
[<oszlopkifejezés> [NOT] IN <értéklista>]
```

A kifejezés hatására annak vizsgálata történik meg, hogy az <oszlopkifejezés> értéke szerepel-e a megadott értéklistában, vagy nem. Amennyiben szerepel a kifejezés értéke igaz lesz. Amennyiben használjuk a NOT kulcsszót, a kifejezés akkor lesz igaz, ha az értéke nem szerepel a listában. Az <értéklista> paraméterben tehát a kifejezés típusának megfelelő értékeket kell vesszővel elválasztva felsorolni.

A predikátum kiemelt szerepet kap a beágyazott lekérdezéseknél.

A harmadik fajta predikátum karakterlánc típusú kifejezésre alkalmazható. Általános formája az alábbi:

```
[<oszlopkifejezés> LIKE <karakterlánc>]
```

A <karakterlánc> konstansban idézőjelek között adhatunk meg karaktersorozatot.

A karaktersorozatban két karakternek speciális jelentése van, ezek a % illetve az _ jelek.

Az <oszlopkifejezés> paraméternek karakteres értéket kell szolgáltatni, amely összehasonlításra kerül a konstanssal. Amennyiben a kettő megegyezik, a feltétel igaz lesz. Ha a konstansban a % jelet használjuk, akkor a két karakterláncnak csak eddig a jelig kell egyezni ahhoz, hogy a feltétel igaz legyen. Ennek megfelelően a % tetszőleges számú karaktert helyettesít. Az _ jel pontosan egy karaktert helyettesít.

7. példa

Készítsünk olyan táblázatot, amely megadja a vállalat azon dolgozóinak nevét és fizetését, akik 150000 Ft felett keresnek!

A feladatot az alábbi paranccsal oldhatjuk meg:

```
SELECT Neve, Fizetes FROM Dolgozo
WHERE Fizetes >= 150000
```

A keletkezett eredménytábla a következő:

Neve	Fizetes
Nagy József	156000
Gipsz Jakab	210000
Kovács Piroska	189000

8. példa

Készítsünk olyan táblázatot, amely megadja a vállalat azon dolgozóinak nevét és fizetését, akik Egerben laknak és a fizetésük 150 000 Ft felett van!

A feladatot az alábbi paranccsal oldhatjuk meg:

```
SELECT Neve, Fizetes FROM Dolgozó
WHERE (Varos = 'Eger') AND (Fizetes >= 150000)
```

A keletkezett eredménytábla a következő:

Neve	Fizetes
Nagy József	156000

9. példa

Készítsünk olyan táblázatot, amely megadja a vállalat azon dolgozóinak nevét és fizetését, akiknek a fizetése 150000 és 200000 Ft közé esik!

A feladatot a következő módon oldhatjuk meg:

```
SELECT Neve, Fizetes FROM Dolgozó
  WHERE Fizetes BETWEEN 150000 AND 200000
```

A feladatot megoldhatjuk másképpen is:

```
SELECT Neve, Fizetes FROM Dolgozó
  WHERE (Fizetes >= 150000) and (Fizetes <= 200000)
```

A keletkezett eredménytábla a következő:

Neve	Fizetes
Nagy József	156000
Kovács Piroska	189000

10 példa

Készítsünk olyan táblázatot, amely megadja a vállalat azon dolgozóinak nevét és városát akik Egerben, Szegeden vagy Debrecenben laknak!

A feladatot a következő módon oldhatjuk meg:

```
SELECT Neve, Varos FROM dolozo
WHERE Varos IN („Eger”,”Szeged”,”Debrecen”)
```

A keletkezett eredménytábla a következő:

Neve	Varos
Kiss Szilárd	Eger
Nagy József	Eger

11. példa

Készítsünk olyan táblázatot, amely megadja a vállalat Kovács nevű dolgozóinak adatait!

A feladatot a következő módon oldhatjuk meg:

```
SELECT * FROM Dolgozo WHERE Neve LIKE "Kovács%"
```

A keletkezett eredménytábla a következő:

ID	Neve	Szdatum	Varos	Fizetes
17	Kovács Piroska	1996.07.15	Budapest	189000

Csoportosító lekérdezések, a group by és a having használata, rendezés

A csoportosítás azt jelenti, hogy a rekordokat egy vagy néhány adott mező értékei szerint csoportokra bontjuk. Ezután a csoportokhoz tartozó rekordokra különböző műveleteket hajthatunk végre, például alkalmazhatjuk a már ismert aggregációs függvényeket, esetleg a csoportokra vonatkozóan kiválasztó műveletet alkalmazhatunk.

A csoportosítás a következő utasítással hajtható végre:

```
GROUP BY <oszlopnév>,[<oszlopnév>]…
```

A csoportosítás a megadott oszlopnevek azonos értékei alapján fog történni. Amennyiben több oszlopot adunk meg, akkor az első oszlop azonos értékein belül a második oszlop azonos értékei szerint csoportosít, majd a harmadik szerint, stb. Az egyes mezőkhöz tartozó értékeket a megadás sorrendjében egymás mellé rakja a rendszer, és az így kapott minta alapján csoportosít.

A művelet végrehajtása után minden egyes csoportra egy sor keletkezik az eredménytáblában.

Mivel speciális utasításról van szó, használata során a SELECT parancs egyéb részeire vonatkozóan is megkötéseket kell tennünk.

Így a lekérdezendő oszlopok adataira vonatkozóan mindenképpen alkalmaznunk kell valamilyen aggregáló operátort, vagy ha ezt nem tesszük, akkor az oszlopnak szerepelnie kell a csoportosításban részt vevő oszlopok között, azaz a GROUP BY után.

Az SQL nyelv lehetőséget biztosít arra is, hogy az aggregálással keletkezett adatokra vonatkozóan feltételeket adhassunk meg. Ebben az esetben a HAVING szót kell használnunk.

Az alparancs formája az alábbi:

```
HAVING <csoportfeltétel>
```

A <csoportfeltétel> paraméterben a hagyományos módon adhatunk meg feltételeket, azzal a különbséggel, hogy a feltételben szereplő oszlopneveknek tartalmazniuk kell valamilyen aggregáló operátort, és ennek ugyancsak szerepelnie kell a SELECT után.

Az SQL lehetőséget biztosít arra, hogy lekérdezéseink eredményét rendezetten jelenítsük meg. ORDER BY alparancs. Formája a következő:

```
ORDER BY <oszlopnév|oszlopsorszám> [ASC|DESC], <oszlopnév| oszlop-
sorszám> [ASC|DESC]]…
```

A rendezés a megadott oszlopok szerint történik. Első szempontként az első oszlopot, további szempontként az utána megadott oszlopokat veszi figyelembe. Az oszlopok kétféleképpen adhatók meg.
Egyrészt hagyományosan a nevükkel, másrészt egy számmal, ami a táblázatban az oszlop sorszáma. A számozás 1-től kezdődik a táblázat fejrészében megadott sorrend szerint.
Lényeges hogy a rendezési szempontként megadott oszlopnak szerepelni kell a SELECT parancs után is. Fontos még az ASC és a DESC kulcsszavak jelentése.
Az ASC az alapértelmezés, ami azt jelenti, hogy a rendezés növekvő sorrend szerint történik.
Amennyiben a DESC kulcsszót használjuk, akkor a megadott szempontnál a rendezés csökkenő lesz.
A következőkben nézzünk néhány példát a csoportosító, illetve a rendező SQL parancsok használatára.

12. példa
Készítsünk olyan listát, amely megadja városonként, hogy az adott városban hány dolgozó lakik!
Az alábbi parancsot használhatjuk:

```
SELECT Varos, COUNT(id) as „DB”  FROM dolgozo
GROUP BY Varos
```

Eredményképpen a következő táblát kapjuk:

Varos	**DB**
Eger	2
Miskolc	1
Budapest	1

13. példa

Készítsünk olyan listát, amely megadja városonként, hogy az ott lakó dolgozóknak mennyi az összes illetve az átlagfizetése!

Az alábbi parancsot használhatjuk:

```
SELECT varos, sum(Fizetes) as „Összes”, avg(Fizetes) as „Átlag”
FROM dolgozo
GROUP BY Varos
```

Eredményképpen a következő táblát kapjuk:

Varos	Összes	Átlag
Eger	276000	138000
Miskolc	210000	210000
Budapest	189000	189000

14. példa

Készítsünk olyan listát, amely megadja azokat a városokat, amelyekre igaz, hogy az adott városban élő dolgozóknak az átlagfizetése legfeljebb 150000 Ft!

Az alábbi parancsot használhatjuk:

```
SELECT varos, avg(Fizetes) as „Átlag”
FROM dolgozo
GROUP BY Varos
HAVING avg(Fizetes) <= 150000
```

Eredményképpen a következő táblát kapjuk:

Varos	Átlag
Eger	138000

15. példa

Készítsünk olyan listát, amely a dolgozók nevét és fizetését a fizetés szerint csökkenő, illetve azonos fizetés esetén név szerint ábécé sorrendben adja meg!

Az alábbi parancsot használhatjuk:

```
SELECT Neve, Fizetes FROM dolgozo
ORDER BY Fizetes DESC, Neve
```

Eredményképpen a következő táblát kapjuk:

Neve	Fizetes

Gipsz Jakab	210000
Kovács Piroska	189000
Nagy József	156000
Kiss Szilárd	120000

Táblák összekapcsolása

A legtöbb esetben az adatbázis szerkezete olyan, hogy a szükséges információk több táblában találhatók. Különösen igaz ez, ha normalizált relációkkal dolgozunk, hiszen mint láttuk, a normalizálás alaptevékenysége a több relációra bontás. Ilyen esetekben az információk összegyűjtéséhez minden táblára szükségünk van, ezért a lekérdezéseket ki kell terjeszteni úgynevezett többtáblás lekérdezésekké.
Az SQL nyelv lehetőséget biztosít az összekapcsolás relációalgebrai művelet közvetlen megvalósítására. Ez azt jelenti, hogy speciális utasítások állnak rendelkezésre, amelyek az összekapcsolás különböző fajtáit adják meg.
A legelső ilyen parancs magát a Descartes szorzatot hozza létre. A parancs megadásánál csak a FROM kulcsszó utáni részt definiáljuk:

```
<táblanév> CROSS JOIN <táblanév>
```

A parancs hatására a megadott két tábla Descartes szorzatát képezi a rendszer. Mint tudjuk, sokkal természetesebb a feltételen alapuló összekapcsolás.
Ennek formája az alábbi:

```
<táblanév> INNER JOIN <táblanév> ON <feltétel>
```

Feltételként tetszőleges, a WHERE parancs után használható feltételt adhatunk meg. Esetek legnagyobb részében a két kapcsolt táblából az egyik tábla (Nevezzük master-nek) elsődleges kulcs mezőjének értéke egyenlő a másik tábla (nevezzük detail-nek) idegen kulcs mezőjének értékével. Mivel a relációs adatbázis kezelők nem kezelik közvetlenül a több-több kapcsolatot, így ez a megfeleltetés egyértelmű.
Ezt az összekapcsolást szoros belső kapcsolatnak is nevezzük.
Tekintsük az alábbi Kifizetes táblát, ami a dolgozóknak történt kifizetések adatait tartalmazza. Ez egy – több kapcsolatban van a már ismert Dolgozo táblánkkal.

Dolgozo				
ID	**Neve**	**Szdatum**	**Varos**	**Fizetes**
12	Kiss Szilárd	1985.01.05	Eger	120000
13	Nagy József	1973.05.06	Eger	156000
15	Gipsz Jakab	1955.11.01	Miskolc	210000
17	Kovács Piroska	1996.07.15	Budapest	189000

Kifizetes

ssz	Datum	Osszeg	Dolgozo
101	2012.01.05	120000	12
102	2012.01.05	160000	13
103	2012.01.05	200000	15
104	2012.02.05	123000	12
105	2012.02.05	240000	15

A két tábla között a kapcsolatot a Dolgozo tábla ID elsődleges kulcsa és a Kifizetes tábla dolgozo idegen kulcs mezője tartja.

16.példa

Készítsünk olyan listát, amely a dolgozók nevét és a számukra kifizetett összeget, és a kifizetés dátumát név szerint ábécé sorrendben adja meg!

Az alábbi parancsot használhatjuk:

```
SELECT dologzo.neve as „Név”,
  kifizetes.osszeg as „Összeg”,
  kifizetes.datum as „Dátum”
FROM dolgozo INNER JOIN kifizetes ON dolgozo.id = kifizetes.dolgozo
ORDER BY 1
```

A keletkezett eredménytábla az alábbi:

Név	Összeg	Dátum
Gipsz Jakab	200000	2012.01.05
Gipsz Jakab	240000	2012.02.05
Kiss Szilárd	120000	2012.01.05
Kiss Szilárd	123000	2012.02.05
Nagy József	160000	2012.01.05

Látható, hogy a Kovács Piroska nevű dolgozó nem kapott még kifizetést, nem is került be az eredmény táblába. Amennyiben szeretnénk azokat a rekordokat is megjeleníteni, melyekhez nincs kapcsolt rekord a kapcsolt táblában külső összekapcsolást kell alkalmazni.

Ezek tulajdonképpen annak a problémának a kezelésére használhatók, ami akkor jelentkezik, ha a két összekapcsolandó tábla valamely sorához nem tartozik a másik táblából elem. Ilyenkor az összekapcsolás művelet definíciója alapján ez a sor nem kerül be az eredménytáblába. A gyakorlatban előfordulhat, hogy ezekre a "lógó" előfordulásokat is szeretnénk szerepeltetni az összekapcsolt relációban. Ilyenkor használhatjuk a külső összekapcsolásokat. Egy külső összekapcsolás abban különbözik a hagyományostól, hogy az eredménybe minden olyan sor is bekerül, amely a másik tábla egyetlen sorához sem kapcsolódik. Egy külső összekapcsolás háromféle módon valósulhat meg. Az egyik mód az, amikor mindkét tábla "lógó" sorai bekerülnek az eredménybe, a másik kettő pedig amikor csak a bal, illetve a jobboldali tábláé.

Ennek megfelelően a következő esetek lehetnek:

```
<táblanév> FULL OUTER JOIN <táblanév> ON <feltétel>
```

Ebben az esetben mindkét tábla "lógó" sorai bekerülnek az eredmény táblába.

```
<táblanév> LEFT OUTER JOIN <táblanév> ON <feltétel>
```

Ebben az esetben csak az első <táblanév> paraméterben megadott tábla "lógó" sorai kerülnek be az eredmény táblába.

```
<táblanév> RIGHT OUTER JOIN <táblanév> ON <feltétel>
```

Ebben az esetben pedig a második <táblanév> paraméterben megadott tábla "lógó" sorai kerülnek be az eredmény táblába.

17. példa

Készítsünk olyan listát, amely az előző példában szereplő feladatot úgy oldja meg, hogy a listába azok a dolgozók is belekerülnek, akiknek nem történt kifizetés.

A megoldás a következő:

```
SELECT dologzo.neve as „Név”,
  kifizetes.osszeg as „Összeg”,
  kifizetes.datum as „Dátum”
FROM dolgozo LEFT OUTER JOIN kifizetes
  ON dolgozo.id = kifizetes.dolgozo
ORDER BY 1
```

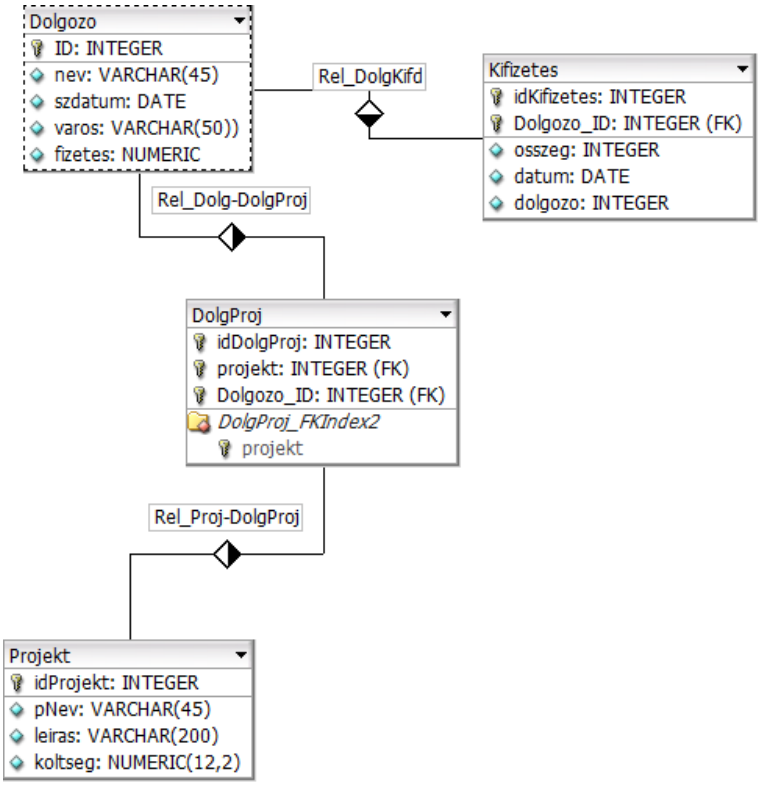
A keletkezett eredménytábla az alábbi:

Név	Összeg	Dátum
Gipsz Jakab	200000	2012.01.05
Gipsz Jakab	240000	2012.02.05
Kiss Szilárd	120000	2012.01.05
Kiss Szilárd	123000	2012.02.05
Nagy József	160000	2012.01.05
Kovács Piroska	null	null

Előfordulhat, hogy esetleg több táblát kell összekapcsolni. Ekkor is teljesen hasonlóan járunk el. Minden összekapcsolás egy JOIN parancs lesz.

Tehát ha 5 táblát szeretnénk összekapcsolni, akkor 4 JOIN lesz a lekérdezésünkben.

Tekintsük a dolgozók részvételét a céges projektekben. Ez több-több kapcsolat, így egy kapcsolótáblára is szükség lesz. Ebben az esetben így néz ki az adatbázisunk:



18. példa

Listázzuk ki a projektek neveit a résztvevő dolgozók neveivel rendezetten.

A megoldás a következő:

```
SELECT p.pNev, d.Nev
FROM dolgozo d
INNER JOIN DolgProj dp ON d.id = dp.dolgozo
 INNER JOIN projekt p ON dp.projekt = p.idProjekt
ORDER BY p.pNev, d.nev
```

Látható, hogy alkalmaztuk a táblák nevének rövidítését. A dolgozo tábla d-vel lett jelölve. Ennek célja a rövidebb írásmód a lekérdezésen belül. Szabály, hogy ha egy tábla nevét rövidítjük, akkor a teljes lekérdezésben a rövidített nevet kell használni.

Beágyazott lekérdezések

Az SQL nyelv lehetőséget biztosít arra, hogy a kiválasztó lekérdezések feltételében is használjunk SQL lekérdező parancsot. Ilyenkor megkülönböztetünk külső és belső SELECT parancsot, és az ilyen jellegű lekérdezéseket beágyazott lekérdezéseknek nevezzük.
A SELECT parancsok szintaxisa megegyezik a már ismert formákkal, csupán arra kell ügyelnünk, hogy a belső lekérdezésekre vonatkozóan bizonyos megkötéseknek kell teljesülnie. A belső lekérdezés jellege alapján több esetet különböztethetünk meg.
Ezek a következők:

1. A belső lekérdezés egyetlen értéket szolgáltat. Ez a legegyszerűbb eset, ugyanis ilyenkor minden a hagyományos módon történik, azzal a különbséggel, hogy a feltételként megadott kifejezésben a belső lekérdezés által szolgáltatott értéket használja fel a rendszer.

2. A belső lekérdezés egyoszlopos relációt szolgáltat. Ekkor olyan feltételeket adhatunk meg, amelyek a belső lekérdezés által szolgáltatott oszlop adatait használja fel. Ebben az esetben különböző predikátumokat használhatunk.

Az alábbi három predikátum létezik:

```
<oszlopkifejezés> [NOT] IN <belső lekérdezés>
```

Ennél a típusnál az <oszlopkifejezés> paraméterben megadott kifejezés értékéről fogja eldönteni a rendszer, hogy szerepel-e a belső lekérdezés által előállított oszlop adatai között. Ha igen a feltétel értéke igaz, ha nem, akkor hamis lesz. A NOT kulcsszó hatására pontosan fordítva kell értelmezni a feltételt.
A következő predikátumok formája az alábbi:

```
[NOT] <oszlopkifejezés> <reláció> ALL|ANY <belső lekérdezés>
```

Ennél a típusnál az <oszlopkifejezés> paraméterben megadott kifejezés értékére vonatkozóan azt fogja vizsgálni a rendszer, hogy a megadott reláció teljesül-e a belső lekérdezés által előállított oszlop adataira. Ha az ALL kulcsszót használjuk, a feltétel akkor lesz igaz, ha a reláció az oszlop minden elemére teljesül, míg az ANY használatakor elegendő egyetlen elemre teljesülnie. A NOT kulcsszó itt is a feltétel ellentettjét jelenti.

A legáltalánosabb eset az, amikor a belső lekérdezés általános relációt szolgáltat. Ekkor csak kétféle feltételt vizsgálhatunk.

Az egyik az, hogy a keletkezett reláció üres vagy sem. Ehhez a vizsgálathoz az EXISTS kulcsszót kell használnunk, amit természetesen a NOT módosíthat. A parancs formája az alábbi:

```
[NOT] EXISTS <belső lekérdezés>
```

A másik eset hasonló az 2. pont első részében leírtakhoz, azzal a különbséggel, hogy a feltételben több oszlopkifejezést adhatunk meg, amelyek egyezését külön külön fogja vizsgálni a rendszer a belső lekérdezés által adott tábla sorainak elemeivel. Természetesen a megadott lista és a lekérdezés eredménytáblája sorainak száma azonos kell hogy legyen. A pontos szintaxis a következő:

```
(<oszlopkifejezés> [,<oszlopkifejezés>]…) [NOT] IN <belső lekérdezés>
```

19. példa

Készítsünk olyan listát, amely a Dolgozó táblából kilistázza azon dolgozók nevét és fizetését, akik az átlag alatt keresnek!

```
SELECT Neve, Fizetes FROM Dolgozo
WHERE Fizetes <
(SELECT AVG(Fizetes) FROM Dolgozo)
```

A keletkezett eredménytábla az alábbi:

Neve	Fizetes
Kiss Szilárd	120000
Nagy József	156000

20. példa

Készítsünk olyan listát, amely a Dolgozó táblából kilistázza azon dolgozók nevét és fizetését, akik a fizetése a legnagyobb fizetéstől legfeljebb csak 50 000 Ft-tal tér el!

Az alábbi parancsot használhatjuk:

```
SELECT Neve, Fizetes FROM Dolgozo
WHERE Fizetes+50000 >
(SELECT MAX(Fizetes) FROM Dolgozo)
```

A keletkezett eredménytábla az alábbi:

Neve	Fizetes

Kiss Szilárd	120000
Nagy József	156000
Gipsz Jakab	210000
Kovács Piroska	189000

21. példa

Az előzőekben látott Kifizetés tábla felhasználásával készítsünk olyan listát, amely a Dolgozó táblából kilistázza azon dolgozók nevét és törzsszámát, akik számára még nem történt kifizetés!

Az alábbi parancsot használhatjuk:

```
SELECT Neve, ID FROM Dolgozo
WHERE ID NOT IN
(SELECT ID FROM Kifizetes)
```

Egy másik lehetséges megoldás:

```
SELECT Neve, ID FROM Dolgozo
WHERE NOT EXISTS
(SELECT ID FROM Kifizetes
WHERE Kifizetes.dolgozo = Dolgozo.ID)
```

Figyeljük meg, hogy ez a második megoldás egy olyan speciális esetet foglal magába, amikor a belső lekérdezésben felhasználjuk a külső lekérdezés táblájának egy mezőjét is.

A keletkezett eredménytábla az alábbi:

Neve	**ID**
Kovács Piroska	17

22. példa

Készítsünk olyan listát, amely a Dolgozó táblából kilistázza azon dolgozók adatait, akik minden Egerben vagy Szegeden lakó dolgozónál többet keresnek!

Az alábbi parancsot használhatjuk:

```
SELECT * FROM Dolgozo
WHERE Fizetes > ALL
(SELECT Fizetes FROM Dolgozo
WHERE Varos = "Eger" OR Varos = "Szeged")
```

ID	**Neve**	**Szdatum**	**Varos**	**Fizetes**

15	Gipsz Jakab	1955.11.01	Miskolc	210000
17	Kovács Piroska	1996.07.15	Budapest	189000

Feladatok

1. Alkotás – Alkotó

Az alkotásról tudjuk, hogy melyik évben készült, mi a műfaja (szobor, festmény, rajzkönyv stb) és mennyi az értéke forintban, ismerjük a címét, a művészt, a művész születési évét, a nemzetiségét. Egy alkotás mindig egy műfajhoz tartozik, és a művészek is egy nemzetnek fiai. A művészek akár teljes munkásságát is tudjuk feldolgozni!

Leadandó az adatbázis terv táblanevekkel, mezőnevekkel, mezőtípusokkal, kapcsolatokkal rajzban.

A táblákat létrehozó DDL parancsok.

Az adatfeltöltést végző DML utasítások.

A válaszokat adó QL, és DDL, DML parancsok.

1. Tervezzen minimum 3NFben lévő adatbázist a fenti probléma kezelésére!

2. Vegyen fel minimum 5 alkotót, és minimum 10 alkotást. Legyen legalább 3 féle műfaj és nemzet is. Töltse fel a megfelelő sorrendben a létrehozott adattáblákat.

3. Válaszoljon SQL lekérdezésekkel az alábbi kérdésekre.

a. Listázza ki a XX. Századi alkotások címeit,a szerző nevével, Szerző neve alapján növekvő sorrendben.

b. Listázza ki a szobrok minden adatát.

c. Listázza ki a magyar művészek adatait névsor szerint csökkenő sorrendben.

d. Határozza meg nemzetenként az alkotások összes értékét.

e. Listázza ki az olasz alkotók alkotásainak átlagértékénél magassabb értékű alkotásokat az alkotók neveivel.

f. Listázza ki a műfajok neveit, de mindegyik csak egyszer szerepelhet.

g. Listázza ki nemzetenként az összes alkotás darabszámát a nemzetek nevével, de csak a 3nál több alkotást jegyzők jelenjenek meg!

h. Vegyen fel a megfelelő táblába egy plusz mezőt, mely az alkotó nemzetközi díjainak számát tartalmazza.

Megoldás az SQL lekérdezésekre:

A.

```
select al.cim, sz.nev
from alkotas al
 inner join K1 on al.al_id = K1.al_id
 inner join alkoto sz on k1.a_id = sz.a_id
where  al.al_ev between 1901 and 2000
order by sz.nev;
```

A/2

```
select al.cim, sz.nev
from alkotas al, K1, alkoto sz
where
(al.al_id = K1.al_id) and
(k1.a_id = sz.a_id) and
(al.al_ev between 1901 and 2000)
order by sz.nev;
```

B.

```
select sz.nev, al.*, n.n_nev
from alkotas al
 inner join K1 on al.al_id = K1.al_id
 inner join alkoto sz on k1.a_id = sz.a_id
 inner join nemzet n on n.n_id = sz.n_id
 inner join mufaj m on m.m_id = al.m_id
where m.m_nev = "szobor"
order by al.al_nev
```

C.

```
select sz.*
from alkoto sz
 inner join nemzet n on n.n_id = sz.n_id
where n.n_nev = "magyar"
order by sz.a_nev desc;
```

D.

```
select n.n_nev as nemzet, sum(al.al_ertek) as osszes
from alkotas al
 inner join K1 on al.al_id = K1.al_id
 inner join alkoto sz on k1.a_id = sz.a_id
 inner join nemzet n on n.n_id = sz.n_id
group by n.n_nev;
```

E.

```
select al.al_nev, sz.a_nev
from alkotas al
 inner join K1 on al.al_id = K1.al_id
 inner join alkoto sz on k1.a_id = sz.a_id
where
al.al_ertek >
 (
 select avg(al.al_ertek)
 from alkotas al
 inner join K1 on al.al_id = K1.al_id
 inner join alkoto sz on k1.a_id = sz.a_id
 inner join nemzet n on n.n_id = sz.n_id
 where n.n_nev = "olasz"
 )
order by sz.a_nev;
```

F/B

```
select m.m_nev
from mufaj m
where m.m_id not in
(
 select distinct m.m_id
 from mufaj m
       inner join alkotas al on m.m_id = al.m_id
)
order by m.m_nev;
```

G.

```
select n.n_nev as nemzet, count(al.al_id) as szamossag
from alkotas al
 inner join K1 on al.al_id = K1.al_id
 inner join alkoto sz on k1.a_id = sz.a_id
 inner join nemzet n on n.n_id = sz.n_id
group by n.n_nev
having count(al.al_id) > 3
```

2.

Alaptáblák

ember [id integer primary key, nev varchar(40) not null, varos varchar(40)]

auto [rsz char(7) primary key, tulaj int, tipus varchar(20), szin varchar(20), ar numeric(7,0)]
Feladatok:

1. Kérdezzük le a piros színű autók árait.
2. Az 500000 és 1000000 Ft közötti értékű autók árát növeljük 20%-al.
3. Kérdezzük le a 'K' betűvel kezdődő tulajdonosokat és autóik típusát.
4. Kérdezzük le a miskolci és egri autótulajdonosok nevét és autóik árát, a tulajdonosok szerint névsorrendbe rendezve.
5. Kérdezzük le azoknak az autótulajdonosoknak a nevét, akiknek 1 millió Ft-nál olcsóbb autójuk van.
6. Kérdezzük le azon autótulajdonosoknak a nevét és címét, akiknek van autójuk.
7. Azoknak az autóknak a rendszámát kérdezzük le, amelyeknek miskolci a tulajdonosuk.
8. Kérdezzük le azokat az autókat, amelyeknek az ára nagyobb minden piros autóénál.
9. Kérdezzük le a miskolci autók átlagárát.
10. Kérdezzük le, hogy az egyes városokban hány autó van.
11. Kérdezzük le az átlagárnál drágább autók rendszámát és tulajdonosaik nevét.
12. Kérdezzük le azon autótulajdonosok számát lakóhelyük szerint csoportosítva és rendezve, akiknek egy és csakis egy autójuk van.

Megoldás az SQL lekérdezésekre:

1. Kérdezzük le a piros színű autók árait.

```
SELECT ar FROM auto WHERE szin = 'piros';
```

2. Az 500000 és 1000000 Ft közötti értékű autók árát növeljük 20%-al.

```
UPDATE auto SET ar=ar*1.2 WHERE ar BETWEEN 500000 AND 1000000
```

3. Kérdezzük le a 'K' betűvel kezdődő tulajdonosokat és autóik típusát.

```
SELECT e.nev, a.tipus
FROM ember e inner join
       auto a on e.id = a.tulaj
WHERE e.nev LIKE 'K%';
```

4. Kérdezzük le a miskolci és egri autótulajdonosok nevét és autóik árát, a tulajdonosok szerint névsorrendbe rendezve.

```
SELECT ember.nev, auto.ar
FROM ember e inner join
       auto a on e.id = a.tulaj
WHERE  varos IN ('Miskolc', 'Eger')
ORDER BY ember.nev;
```

5. Kérdezzük le azoknak az autótulajdonosoknak a nevét, akiknek 1 millió Ft-nál olcsóbb autójuk van.

```
SELECT ember.nev
FROM ember e inner join
       auto a on e.id = a.tulaj
WHERE  auto.ar < 1000000;
```

6. Kérdezzük le azon autótulajdonosoknak a nevét és címét, akiknek van autójuk.

```
SELECT nev, varos
FROM ember
WHERE id IN (SELECT tulaj FROM auto);
```

7. Azoknak az autóknak a rendszámát kérdezzük le, amelyeknek miskolci a tulajdonosuk.

```
SELECT a.rsz
FROM  ember e inner join
       auto a on e.id = a.tulaj
WHERE  e.varos LIKE 'Misk%';
```

8. Kérdezzük le azokat az autókat, amelyeknek az ára nagyobb minden piros autóénál.

```
SELECT *
FROM auto
WHERE ar >
 (SELECT max(ar) FROM auto WHERE szin LIKE 'pir%');
```

9. Kérdezzük le a miskolci autók átlagárát.

```
SELECT AVG(ar)
FROM ember e inner join
       auto a on e.id = a.tulaj
WHERE  varos LIKE 'Mi%';
```

10. Kérdezzük le, hogy az egyes városokban hány autó van.

```
SELECT varos, COUNT(*)
FROM ember e inner join
       auto a on e.id = a.tulaj
```

```
GROUP BY varos;
```

11. Kérdezzük le az átlagárnál drágább autók rendszámát és tulajdonosaik nevét.

```
SELECT auto.rsz, ember.nev
FROM ember e inner join
       auto a on e.id = a.tulaj
WHERE  ar > (SELECT AVG(ar) FROM auto);
```

12. Kérdezzük le azon autótulajdonosok számát lakóhelyük szerint csoportosítva és rendezve, akiknek egy és csakis egy autójuk van.

```
SELECT varos, COUNT(*)
FROM ember
WHERE nev IN (SELECT nev
FROM ember e inner join
       auto a on e.id = a.tulaj
 GROUP BY nev
 HAVING COUNT(*)=1)
GROUP BY varos
ORDER BY varos;
```

3.

Egy videó kölcsönző

Egy video kölcsönző nyilvántartását vizsgáljuk. A kölcsönző kizárólag videofilmeket kölcsönöz, a videofilmekről nyilvántartást vezetnek, mely nyilvántartás tartalmazza a film címét, főszereplőjét, rendezőjét. A filmeknek egyedi azonosító száma van, a cím nem biztos, hogy egyedi. Egy film több kazettán is szerepelhet, a kölcsönző által készített másolatokban. A kazettáknak szintén egyedi azonosítójuk van, valamint a típusát is nyilvántartják. A kölcsönzőből kizárólag tagok kölcsönözhetnek. A tag nevét, címét, valamint egyedi sorszámát nyilvántartják. A tagok filme(ke)t kölcsönöznek, a kikölcsönzés dátumát és a visszahozás dátumát is pontosan követni kell. A kölcsönző egy nagykereskedőtől vásárolja a filmeket, minden rendelésnek van egy száma és a visszakeresés érdekében a rendelés dátumát is nyilvántartják. Egy rendelésen több film is szerepelhet.

A videó kölcsönzőnél található dokumentumok a következő adatokat tartalmazzák a különböző adatlapokon. Ezekből a dokumentumokból kell megalkotni a logikai adatbázis-modellt:

Kazetta száma

Kazetta típusa

Film címe

Film száma

Rendelés száma

Rendelés dátuma

Kölcsönzés száma

Tag sorszáma

Tag neve

Tag címe

Kölcsönzés dátuma

Visszahozás dátuma

Film főszereplője

Film rendezője

Lekérdezések

1. Tudjuk meg azoknak a tagoknak az adatait, akik bár be vannak iratkozva, még nem kölcsönöztek egyetlen filmet sem !
2. Tudjuk meg az összes rendelkezésre álló filmünk főszereplőjét !
3. Állapítsuk meg az összes rendelkezésre álló film címét és rendezőjét !
4. Jelenítsük meg azon filmek és kazetták számát és a film címét, amelyeket még soha nem kölcsönöztek ki !
5. Listázzuk ki a mai kölcsönzők névlistáját!
6. Listázzuk ki azokat a filmeket, amiket ma kell visszahozni!
7. Listázzuk ki azokat a filmeket, amelyekkel már elkéstek!
8. Készítsen filmlistát!

Megoldás az SQL lekérdezésekre:

1.

```
select * from tag
where tag_sorszam not in
 (select tag_sorszam from kolcsonzes)
order by nev
```

2.

```
select d.nev from dolgozo d
 inner join film_dolg fd on d.do_id = fd.do_id
 inner join dolg_tipus dt on dt.dt_id = fd.dt_id
where dt.leiras = "főszereplő"
order by d.nev
```

3.

```
select f.filmcime, d.nev from dolgozo d
 inner join film_dolg fd on d.do_id = fd.do_id
 inner join dolg_tipus dt on dt.dt_id = fd.dt_id
 inner join film f on f.filmszama = fd.filmszama
where dt.leiras = "rendező"
order by d.nev
```

4.

```
select f.filmcime, k.kazettaszama
from film f
 inner join kazetta k on f.filmszama = k.filmszama
where k.kazettaszama not in
(select disctinct k2.kazettaszama from kazetta k2
 inner join kolcsonzes ko
 on k2.kazettaszama = ko.kazettaszama)
```

5.

```
select t.nev from tag t inner join kolcsonzes k
 on k.tag_sorszama = t.tag_sorszama
where kolcsonzes_datuma = '2005.11.21'
```

6.

```
select f.filmcime, k.kazettaszama
from film f
 inner join kazetta k on f.filmszama = k.filmszama
 inner join kolcsonzes ko
       on k.kazettaszam = ko.kazettaszama
where ko.visszahozas_datuma = '2005.11.21'
order by f.filmcime
```

7.

```
select f.filmcime, k.kazettaszama
from film f
 inner join kazetta k on f.filmszama = k.filmszama
 inner join kolcsonzes ko
       on k.kazettaszam = ko.kazettaszama
where ko.visszahozas_datuma < '2005.11.21'
order by f.filmcime
```

8.

```
select filmcime from film order by filmcime
```

4. feladat

Királyok

A magyar királyok uralkodásának idejét szeretnénk tárolni és feldolgozni. Ismert a királyok neve, az uralkodásuk kezdő és végső éve.

Tábla:

URALKODÓ (AZ, Név, Kezdő, Végső)

AZ A király azonosítója (számláló), ez a kulcs

Név A király neve (szöveg)

Kezdő A király uralkodásának kezdő éve (szám)

Végső A király uralkodásának befejező éve (szám)

1. Listázza ki a királyokat névsorba! (Névsor)
2. Írassa ki a László nevű királyokat az uralkodásuk hosszával együtt! Az Ulászlók ne szerepeljenek a listán! (Lászlók)
3. Listázza ki a királyok nevét uralkodásuk hosszának sorrendjében csökkenően! (HosszSor)
4. Az adatbázisból kérdezze le, hogy hány István keresztnevű uralkodónk volt? (Istvánok száma)
5. 1347-ben a királyi udvar Visegrádról átmenetileg Budára költözött. Ki volt ekkor a király? (Budai udvar)
6. Írassa ki a 10 évnél hosszabb ideig uralkodó királyokat névsorba! (10 év)
7. Hány király volt Magyarországon a XIV. században, azaz 1300-tól 1399-ig? Figyelem, a királyok uralkodásának csak egy része is eshetett a megjelölt időszakba! (Királyok száma)
8. Allekérdezés segítségével határozza meg, hogy Mátyás előtt hány király uralkodott? (Mátyás előtt)
9. Ki uralkodott legtovább? (MaxKirály)

5. feladat

Vízi sporteszközök kölcsönzése

A vízparti nyaralás során kölcsönözni lehet sporteszközöket naponta több órára és több alkalommal is.

Készítsen adatbázis-alkalmazást a kölcsönzőt üzemeltetők számára!

Az adatbázisban tárolja, hogy melyek azok az eszközök, amiket bérbe adnak, továbbá azt, hogy ezek milyen típusúak (pl. vízi-bicikli, csónak, kajak, surf), és mennyibe kerül a bérlés

egy órára. Minden megkezdett óráért fizetni kell. Tárolja az ügyfeleket (kölcsönzőket) is, és azt, hogy melyik ügyfél, mikor, melyik eszközt kölcsönözte ki. A kölcsönzős megbízik az ügyfeleiben, ezért nekik csak a nap végén egyszer, egy összegben kell fizetniük a napi kölcsönzésük után.

Mindehhez négy adattáblát készítsen, melyek a következő attribútumokat és kapcsolatokat tartalmazzák (*-gal jelöljük az elsődleges kulcsokat):

Eszkoztipus *Tipus (szöveges), Ar_orara (szám)

Eszkoz *Eazonosito (szám), Tipus (szöveges), Marka (szöveges)

Ugyfel *Uazonosito (szám), Nev (szöveges), Cím (szöveges), Fizetendo (szám)

Kolcsonzes Uazonosito (szám), Eazonosito (szám), Hany_ora (szám), Mitol (idő), Meddig (idő)

1. Hozza létre az adattáblákat a kapott információk alapján! A meződefiniálás során a mezőméretet optimálisan válassza meg! Az idegen kulcsokat (kapcsolatokat) úgy állítsa be, hogy az adatbázis-kezelő megőrizze a hivatkozási integritást!
2. Illessze be a táblákba a kolcson.xls munkafüzet munkalapjain levő adatokat!
3. Készítsen lekérdezést, ami kiírja a Kolcsonzes tábla tartalmát úgy, hogy az Uazonosito mező helyett az ügyfél nevét, az Eazonosito mező helyett pedig az eszköz típusát és márkáját lássuk!
4. Listázza ki az összes olyan eszköz típusát, ami déli 12 órakor bérbe volt adva!
5. Készítsen lekérdezést, ami megadja az összes olyan eszköztípust (egyszer) amilyen eszközt legalább egyszer, legalább 2 órára kikölcsönöztek!
6. Listázza ki az összes olyan eszköz minden adatát, amit nem kölcsönöztek ki
7. Készítsen lekérdezést, ami kiszámolja, hogy egy-egy ügyfélnek összesen mennyit kell fizetni a nap végén (a napi kölcsönzéseinek darabszáma és időtartama (órák) és a bérelt eszköztípusok árai alapján)!

6. feladat

Verseny

Egy feladatmegoldó verseny eredményeinek nyilvántartására hozzon létre az alábbi adatszerkezetnek megfelelő adatbázist! (*-gal jelöljük az elsődleges kulcsokat)

Táblák:

VERSENYZŐ (VersenyzőAZ, Név, IskolaAZ, Osztály)

*VersenyzőAZ A versenyző azonosítója (szöveg)

Név A versenyző neve (szöveg)

IskolaAZ A versenyző iskolájának azonosítója (szöveg)

Osztály A versenyző évfolyama (szám)

EREDMÉNY (VersenyzőAZ, FelSorszám, Pontszám)

*E_id (szám)

VersenyzőAZ A versenyző azonosítója (szöveg)

FelSorszám A feladat sorszáma (szöveg)

Pontszám Az adott feladatban a versenyző által elért pontszám (szám)

ISKOLA (IskolaAZ, IskolaNév, Irszám, Város, Utca, Házszám)

*IskolaAZ Az iskola azonosítója (szöveg)

IskolaNév Az iskola neve (szöveg)

Irszám A versenyző évfolyama (szám)

Város Az iskola város (szöveg)

Utca Az iskola utcája (szöveg)

Házszám Az iskola házszáma (szöveg)

A táblák közötti kapcsolatok:

ISKOLA – VERSENYZŐ: egy a többhöz típusú.

VERSENYZŐ – EREDMÉNY: egy a többhöz típusú.

Feladatok

1. Hozzon létre egy VERSENY nevű adatbázist a fenti táblaszerkezeteknek megfelelően!
2. Az alábbi mintáknak megfelelően töltse fel adatokkal a táblákat!

Név	VersenyzőAZ	IskolaAZ	osztály
Sziroczák Richárd	1	1	9
Hamvas Mihály	10	1	9
Tassy Gergely	11	1	9
Holicska Ábel	12	2	10
Benkő Zsolt	2	2	9
Bogner Orsolya	3	2	10
Réti Kornél	4	3	10
Susuk Márton	5	4	10
Sipos András	6	5	9
Sipos András	7	3	9
Siska Attila	8	2	9

Név	VersenyzőAZ	IskolaAZ	osztály
Bein Ákos	9	4	10

VersenyzőAZ	FelSorszám	Pontszám
1	I	12
1	II	8
1	III	11
1	IV	3
10	I	2
10	II	5
10	III	0
10	IV	6
11	I	4
11	II	9
11	III	13
11	IV	2
És így tovább		

Minden versenyzőnek négy feladatot kellett megoldani! A feladatokat római számmal jelöljük.

Iskola					
IskolaAZ	IskolaNév	Irszám	Város	Utca	Házszám
1	Berzsenyi Dániel Gimnázium	1133	Budapest	Kárpát u.	49-53
2	Könyves Kálmán Gimnázium	1043	Budapest	Tanoda tér	1
3	Eötvös József Gimnázium	1053	Budapest	Reáltanoda	7
4	Árpád Vezér Gimnázium	3950	Sárospatak	Arany János út	3
5	Táncsics Mihály Gimnázium	5900	Orosháza	Táncsics Mihály u.	2

Készítsen lekérdezéseket a következő kérdések megválaszolására! A feladatok megoldásának nevét a zárójelben levő betűkkel jelölje!

3. Írassa ki a versenyzők neveit és pontjainak összegét emelkedő sorrendben! (A)
4. Határozza meg feladatonként a legmagasabb pontszámot! (B)
5. Melyik iskolából (iskola név és város) hány versenyző indult? (C)
6. Gyűjtse ki azoknak a versenyzőknek a neveit ábécé rendbe, akik valamelyik feladatra 0 pontot kaptak! A felsorolásban egy név csak egyszer szerepelhet! (D)

7. Írassa ki azoknak a nevét, iskolája nevét és címét, akiknek van 10 pontnál magasabb pontszámú feladatuk!
8. Az első feladatra a vidékiek közül ki kapta a legtöbb pontot? Nevét és iskolájának nevét adja meg a lekérdezés! (F)

7. feladat

Termékek

Készítse el a Termékek tábla szerkezetét az alábbi minta alapján

Termékek (Termékkód, Terméknév, Kategóriakód, Méret, Szín, Ár, Raktárkészlet)

Termékkód (számláló)

Terméknév (szöveg)

Kategóriakód (szám)

Méret (szöveg)

Szín (szöveg)

Ár (szám)

Raktárkészlet (szám)

Állítsa be a táblák kulcsait!

Változtassa meg a Termékek tábla Ár mezőjének formátumát úgy, hogy az árak egész pénznemként látszódjanak!

A feladatok megoldásánál ügyeljen a táblák közötti kapcsolatokra!

Feladatok:

Készítse el az alábbi feladatok megoldását! A zárójelben megadott néven mentse az egyes megoldásokat!

1. Melyek azok a termékek és milyen színben illetve árban kaphatók, amelyek L-es méretűek és legalább 5 db van belőlük raktáron? Rendezze a listát úgy, hogy a legnagyobb árú termék legyen elöl! (A)
2. Átlagosan menyibe kerülnek az egyes kategóriákba tartozó termékek? (B)
3. Mekkora jelenleg a különböző színű és méretű pamutnadrágok és a vászonnadrágok raktárkészlet-értéke: A listában szerepeljen a termék neve, mérete, színe, kategóriája és raktárkészlet-értéke névsorrendben! (C)
4. Melyek azok a fekete színű top-ok és milyen méretben illetve árban kaphatók, amelyekből van raktáron? (D)
5. Összesen hány darab kötött pulóver van raktáron? (E)

6. Melyek azok a termékek és mennyi van belőlük raktáron, amelyek ára 5 000 és 10 000 Ft között van? Rendezze a listát raktárkészlet szerint növekvő, azonos raktárkészlet esetén névsorrendbe! (F)

8. feladat
Párbajtőr

Az újkori olimpiák férfi, egyéni párbajtőr dobogós helyezetteinek adatait vizsgálja meg adatbázis-kezelő rendszer segítségével! Ezeket az információkat egy adattáblában tároljuk.

Készítsen új adatbázist parbaj néven!

A tábla szerkezete:

EGYÉNI (Azonosító, Év, Helyszín, Helyezés, Név, Ország)

Azonosító A versenyző azonosítója (számláló), ez a kulcs

Év Az olimpia időpontja (szám)

Helyszín Az olimpiát rendező város (szöveg)

Helyezés A versenyző helyezése a versenyszámban (szám)

Név A versenyző neve (szöveg)

Ország A versenyző országának rövidítése (szöveg)

Feladatok

Készítse el a következő feladatok megoldását!

1. Listázza ki az aranyérmes versenyzők nevét, az olimpia idejét és helyét az időpont alapján növekvően! (A)
2. Kulcsár Győző melyik olimpián és milyen helyezést ért el? (B)
3. Írassa ki a atlantai olimpia érmeseinek nevét és helyezését! (C)
4. A magyarok hány dobogós helyezést értek el? (D)
5. Kik voltak a magyar dobogósok? Mindenki csak egyszer szerepeljen a helyezések számával! (E)
6. Ki nyert többször aranyérmet? (F)
7. Összesítse, hogy melyik hogy melyik ország hány aranyérmet szerzett! (G)
8. Melyik városban rendeztek többször olimpiát? (H)

9. feladat

TANFOLYAM

Adattábla neve: DOLGOZÓ

Dolgozó kód	Dolgozó név	Születési dátum	Irányítószám
2000	Kos Miklós	1966.12.12.	1095
2001	Kiss Emese	1970.10.12.	3515
2002	Zili Zoltán	1980.11.25.	4000
2003	Cserepes Virág	1971.02.18.	2840

Adattábla neve: IRÁNYÍTÓSZÁM

Irányítószám	Város
1095	Budapest
3515	Miskolc
4000	Debrecen
2840	Oroszlány

Adattábla neve: TANFOLYAM

Tanfolyam kód	Kezdés	Tanfolyam név	Díj
1	1998.05.01	Informatika	3000
2	1998.05.01	Angol	6000
3	1998.05.01	Német	6000
4	1998.06.02	Pénzügy	7000
5	1998.07.02	Vállalkozási ismeretek	8000

Adattábla neve: LÁTOGATÁS

Dolgozó kód	Tanfolyam kód
2000	1
2000	5
2001	1
2002	4
2003	1
2003	4

Egy vállalattól a dolgozók különféle tanfolyamra járnak. Egy dolgozó több tanfolyamra is beiratkozhat.

1. Készítse el a fenti adattáblákat, határozza meg a mezőtípusokat!

2. Hozza létre értelemszerűen az elsődleges kulcsokat és hozza létre a táblák közötti megfelelő kapcsolatokat!
3. Készítsen lekérdezést azon dolgozókról, akik informatika tanfolyamra járnak!
4. Hányan járnak informatika tanfolyamra?
5. Készítsen lekérdezést azon tanfolyamokról, akik 4500 Ft és 6500 Ft közötti!
6. Csökkentse 50%-al az összes díjat!
7. Készítsen lekérdezést a tanfolyami hallgatók összesen befizetett díjáról!

10. feladat

Statisztika

Egy Budapest VII. kerületi iskola statisztikai kimutatásaihoz készít adatbázist.

1. Tervezzen adatbázist statisztika néven!
2. A diakok nevű adattáblát úgy tervezze meg, és hozza létre, hogy feltöltése után válaszolni tudjunk az alábbi kérdésekre.
   - Mennyi az osztályban a fiúk száma, lányok száma, magántanulók száma?
   - Mennyi a 14, 15, 16, 17, 18 évesek száma?
   - Kinek van 2, vagy annál több testvére?
   - Hányan laknak az iskola székhelyétől eltérő városban, kerületben, illetve hány „kerületi” diák ját az osztályba?
   - Hány kollégista van?
   - Kik veszik igénybe az iskolában a menzát, tanulószobát?
   - Mennyi azon családok száma, ahol a becsült, egy főre jutó jövedelem kisebb, mint 30 000 Ft?
3. Az adattáblában adjon meg kulcsmezőt!
4. Töltse fel az adattáblát 5 rekorddal, kitalált személyek adataival!

Az adatok alapján válaszoljon a következő kérdésekre, válaszait a kérdés után zárójelben található néven mentse:

5. Kinek van 2, vagy annál több testvére? A diákok nevét, testvéreinek számát és egy főre jutó becsült jövedelmét jelenítse meg a jövedelem szerint növekvő rendben! (testver)
6. Hányan laknak az iskola székhelyétől eltérő városban, kerületben? (mashely)

11. feladat

Nobel-díj

Az adatbázis-kezelő segítségével dolgozza fel a Nobel díjasok adatait!

Készítsen új adatbázist nobel néven!

A tábla a következő szerkezetű legyen:

évszám	a díjazás éve	szám
név	a díjazott nev	szöveg
ország	a díjazott országa	szöveg
díj	a díj fajtája	szöveg

1. Az adatbázisból hiányzik Kertész Imre 2002-es irodalmi Nobel-díja. Egészítse ki az adatbázist!
2. Készítse el a magyar Nobel-díjasok listáját! (magyar)
3. Milyen díjat kapott, mikor és milyen ország színeiben Bárány Róbert? (Bárány)
4. Milyen díjat kapott, mikor és milyen ország színeiben Wigner Jenő? (Wigner)
5. Kik voltak a holland (ország jelölésük: NL) Nobel-díjasok? (NL)
6. Melyik díjban részesültek a legtöbben, és hányan? (dijak)
7. Melyik évben milyen díjat nem adtak ki? (hiany)
8. Mely országok képviselői kaptak csak egyszer Nobel-díjat? (egyszer)
9. Listázza ki, hogy melyik ország képviselői hány díjat kaptak! A listában csak tényleges országok szerepeljenek! A lista az országokat rangsorban mutassa! (rangsor)
10. Listázza ki 1901 és 1939 között évenként és díjanként a díjazottakat és országukat! (1939)

12. feladat

A Stupido-Gigantic GmbH Ltd Kft S.A. főleg kereskedelemmel foglalkozik. Ennek megfelelően rendelések futnak be hozzá vásárlóktól, amelyeket a szállítók révén teljesít. Egy terméket csak egy szállító szállít, és egy szállító csak egy terméket szállít. Nyilvántartjuk a vásárlók pillanatnyi egyenlegét. Egy rendelésen több tétel is szerepelhet, a tétel tartalma a termék megnevezése és a mennyiség. A szállítók, termékek és városok neve egyedi, de a vásárlók neve csak egy városon belül egyedi. Feltehetjük, hogy minden városnak csak egy irányítószáma van.

A tárolt adatok ömlesztve:

SzállítóIrányítóSzám, VásárlóIrányítóSzám, SzállítóNév, RendelésSzáma, Mennyiség, RendelésDátuma, VásárlóVárosnév, Egyenleg, Egységár, Terméknév, SzállítóVárosnév, Vásárlónév, Terméknév.

Feladatok:

1. Készítsük el a Stupido-Gigantic GmbH Ltd Kft. S.A. kereskedelmének logikai adatbázis-modelljét (a 3NF alakig)!

2. Valósítsuk meg a logikai adatmodellt!

3. Valósítsuk meg a következő lekérdezéseket:

- a "gumimaci" nevű terméket szállító szállító városának az irányítószáma
- 1998 évi vásárlóink neve
- ajkai vásárlóink egyenlege
- a legnagyobb egyenlegű vásárló neve és városa
- a leghűségésebb vásárlónk melyik szállítótól vásárolt a legtöbbször?

13. feladat

Alaptáblák

ember [id integer primary key, nev varchar(40) not null, varos varchar(40)]

auto [rsz char(7) primary key, tulaj integer, tipus varchar(20), szin varchar(20), ar numeric(7,0)]

Hozzul létre a két adattáblát, és töltsük fel adatokkal. Mindkét táblában legyen min 10-10 rekord.

Feladatok:

1. Kérdezzük le a piros színű autók árait.
2. Az 500000 és 1000000 Ft közötti értékű autók árát növeljük 20%-al.
3. Kérdezzük le a ’K’ betűvel kezdődő tulajdonosokat és autóik típusát.
4. Kérdezzük le a miskolci és egri autótulajdonosok nevét és autóik árát, a tulajdonosok szerint névsorrendbe rendezve.
5. Kérdezzük le azoknak az autótulajdonosoknak a nevét, akiknek 1 millió Ft-nál olcsóbb autójuk van.
6. Kérdezzük le azon autótulajdonosoknak a nevét és címét, akiknek van autójuk.
7. Azoknak az autóknak a rendszámát kérdezzük le, amelyeknek miskolci a tulajdonosuk.
8. Kérdezzük le azokat az autókat, amelyeknek az ára nagyobb minden piros autóénál.
9. Kérdezzük le, hogy milyen típusú autó fordulnak elő az autó táblában (ismétlődés nélkül).

10. Kérdezzük le a miskolci autók átlagárát.
11. Kérdezzük le, hogy az egyes városokban hány autó van.
12. Kérdezzük le az átlagárnál drágább autók rendszámát és tulajdonosaik nevét.
13. Kérdezzük le a legolcsóbb miskolci autónál drágább autók rendszámát.

Nézetek és indexek

Nézet

A felhasználók egyes csoportjai nem látják a teljes adatbázist, esetleg annak részeit is másként látják mint ahogy azok a koncepciós modell szerint felépülnek.
Ilyen eset amikor egy adott csoport a személyi adatokból nem látja a személy rekord Fizetés mezőjét, egy másik csoport pedig a Születési_időt. A nézetek kialakítása is adatbázis-tervezési feladat. Az adatmodell három szintre bontható:

- Külső szintek: Azt írják le, hogyan látják az egyes felhasználók az adatbázist.
- A középső vagy koncepcionális szint: A teljes adatbázis koncepcionális szerkezetét ábrázolja.
- A belső vagy fizikai szint: Az adatok fizikai elhelyezését és elérési módját írja le.

Az adatbázis függetlenség mint az adatbázis-kezelés egyik legfontosabb követelménye, a koncepcionális és a fizikai szint éles különválasztásának köszönhető. Megkülönböztetünk logikai és fizikai adatfüggetlenséget.

A logikai adatfüggetlenséget a metaadatok biztosítják számunkra, vagyis nemcsak az adatokat,hanem az adatok jellemzőit és az adatcsoportok közötti kapcsolatokat leíró adatokat is tároljuk.
Az adatbázis-kezelő rendszerek (DBMS) a logikai adatfüggetlenséget alaptulajdonságuknál fogva biztosítják. Az adatszerkezet megváltoztatása csak a metaadatokban jelent változást, nem kell a felhasználói programot átírni. Lehetséges koncepcionális szinten változtatásokat végezni anélkül, hogy az adatmodell külső szintjében (nézet/view) változás történne.

A DBMS ugyanakkor biztosítja a fizikai adatfüggetlenséget is. Az adattárolási szerkezetek és a hozzáférési módok - vagyis a fizikai adatbázis - változtatása nem vonja maga után a koncepcionális séma és az alkalmazói programok megváltozását. Például egy indexelést a felhasználó legfeljebb oly módon észlel, hogy bizonyos adataihoz gyorsabban hozzáfér, mint korában.

Minden jól megtervezett, a rétegezési koncepció alapján felépített rendszerben cél az, hogy a rétegek egymástól függetlenül megváltoztathatók, kicserélhetők legyenek, amennyiben a rétegek közötti interfészek közben változatlanok maradnak. Az adatbázis kezelés világában ezt az elvet az adatfüggetlenség elvének nevezzük.
Kétféle adatfüggetlenségről szokás beszélni: a fizikai és a fogalmi adatbázis között értelmezhető fizikai adatfüggetlenségről, illetve a fogalmi adatbázis és a nézetek között értelmezhető logikai adatfüggetlenségről.
A fizikai adatfüggetlenségen azt az elvárást értjük, hogy a fizikai szinten, a fizikai működés sémáiban véghezvitt változások ne érintsék a fogalmi (logikai) adatbázist. Ha ez teljesül (gyakorlatilag mindig), akkor a fizikai adathordozó egy teljesen más paraméterekkel rendelkezőre kicserélhető (pl. meghibásodás, technikai fejlődés, stb. miatt), vagy az állományszervezés módja megváltoztatható anélkül, hogy az adatbázisban bármilyen logikai változás lenne érzékelhető.
Logikai adatfüggetlenségről akkor beszélünk, ha a logikai adatbázis megváltozása nem jár az egyes felhasználásokhoz-felhasználókhoz tartozó nézetek megváltozásával. Ez az elvárás már nem teljesül minden esetben.

Nézetet a következő paranccsal hozunk létre.

CREATE VIEW <nézetnév> [(<oszlopnév1>, …)] AS <lekérdezés>;
A lekérdezésre egyetlen egy megkötés van, mégpedig hogy rendezést nem tartalmazhat.
Ha nem adunk meg oszopneveket, a nézet oszlopai a SELECT után felsorolt oszlopok neveivel azonosak.
Viszont ha a SELECT számított értéket is előállít, akkor meg kell adni az oszlopneveket.
Például:
CREATE VIEW dept_sal (deptno, avg salary) AS SELECT deptno, AVG (sal) FROM emp GPOUP BY deptno;
A nézetek a lekérdezésekben a táblákkal megegyező módon használhatók.

Módosítható nézettáblák

- A nézettáblák módosításával a mögöttük lévő adattáblák tartalma módosul
- Alaphelyzetben nem minden nézettábla módosítható
- A módosíthatóság kétféleképpen érhető el:
  - a nézettábla eleve egyszerűbb szerkezetű

- INSTEAD OF típusú triggert készítünk hozzá

A módosíthatóság szerkezeti feltételei

- A nézettábla definíciója nem tartalmazhat:
  - halmazműveletet
  - a DISTINCT, GROUP BY, ORDER BY kulcsszavakat
  - összesítő függvényt
  - al-lekérdezést
- Nem módosíthatók azok az oszlopok, amiknek értéke egy kifejezésből származik
- Több táblára hivatkozó nézetnél ezen kívül
  - a módosítás csak egy táblát érinthet
  - INSERT és UPDATE esetén a módosított tábla kulcsainak és egyedi értékeinek egyedinek kell lenni a nézetben is.

A nézettábla szerkezetének megváltoztatására a CREATE OR REPLACE VIEW parancsot használjuk.

Nézettábla törlése

- Alakja:
  DROP VIEW nézet-név;
- Példa:
  DROP VIEW budapesti_vevok_vw;
- Az adatok nem törlődnek! Azok az adattáblákban vannak!

Foglaljuk össze hogy miért jók nekünk a nézettáblák:

- Először is Biztonság: csak kiválasztott adatokhoz engednek hozzáférést
  - pl. minden főnök csak a saját beosztottainak az adatait láthatja
  - az adatszótárból is csak nézettáblákat látunk (pl. USER_TABLES, ALL_USERS, stb.) Ez azért jó mert csak azt látja a felhasználó amit valóban látnia kell, semmi egyebet nem.

- Egyszerűsítik a felhasználók által kiadandó utasításokat
  - pl. nem kell a felhasználónak táblákat összekapcsolni.

- Függetlenítik az alkalmazásokat a táblák szerkezetétől
  - pl. az alaptáblákhoz új oszlopok adhatók vagy az oszlopok neve megváltoztatható.
  -
- Növelik a biztonságot
  - pl. bonyolult, többször használt lekérdezések nézettáblaként elmenthetők.

–

- Az adatok különböző felhasználói igények szerint „tálalhatók”
  - pl. a felhasználók csak a nekik szükséges adatokat látják.

Az indexek

Az indexrekordokban található mutatók két féle képen azonosítják a keresett rekordot:

- Mutatót állíthatunk minden egyes adatrekordra, ez a sűrű indexelés, vagy
- Mutatót állíthatunk adatrekordok egy csoportjára, tipikusan az egy blokkban lévőkre. Ez a ritka indexelés.

Ritka indexek:

Az indexrekordok azt határozzák meg, hogy az adatállomány rekordjai melyik blokkban találhatóak. –> egy blokkon belül az adatrekordok szabad rekordoknak tekinthetők.
Ritka indexek estén az adatállományt is rendezetten kell tárolni, azaz egy blokkban kell lennie Az összes olyan adatrekordnak, melynek kulcsa egy intervallumba esik.

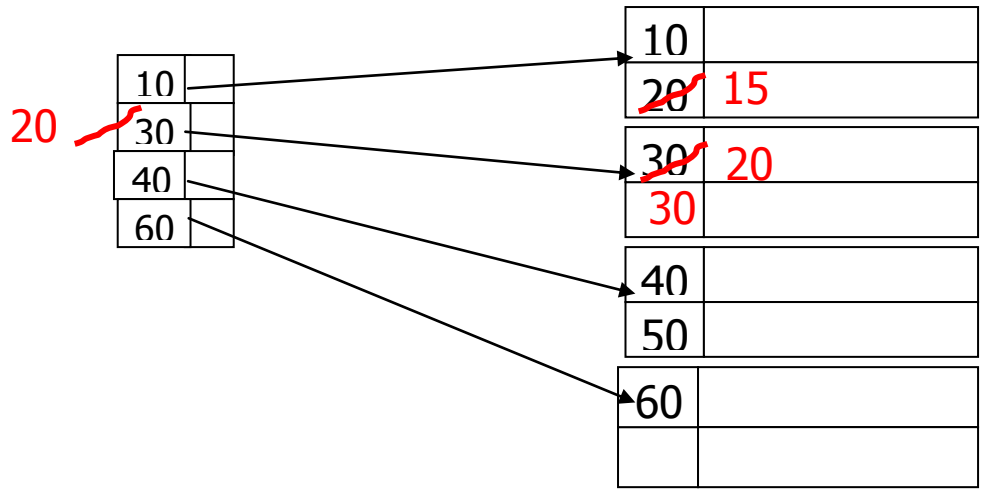
Keresés:

Tegyük fel hogy k1 kulcsú rekordot keressük. Az indexállományban megkeressük azt a rekordot, amelyiknek k2 kulcsa a legnagyobb azok közül, amelyek még kisebbek k1-nél.
A k2 kulcsú indexrekord mutatója megcímzi azt a blokkot, amelyet végig kell keresni a k1 kulcsú adatrekord után.

Beszúrás:

Tegyük fel hogy a k1 kulcsú rekordot tárolni akarjuk. Ehhez meg kell keresnünk az a blokkot, melyben a rekordnak lennie kellene, ha az adatállományban lenne, legyen ez a Bi blokk. Ezután két eset áll fenn:
Vagy van elegendő hely a Bi blokkban a k1 kulcsú rekord számára vagy nincs.
Ha van akkor nincs más dolgunk mint beírjuk a rekordot a Bi blokkba.
Ha nincs akkor helyet kell neki csinálni → kérünk egy új üres blokkot.

Vigyük be a 15-ös rekordot!



- Azonnal újrarendeztük az állományt.
- Másik változatban túlcsordulási blokkot láncolunk a blokkhoz.

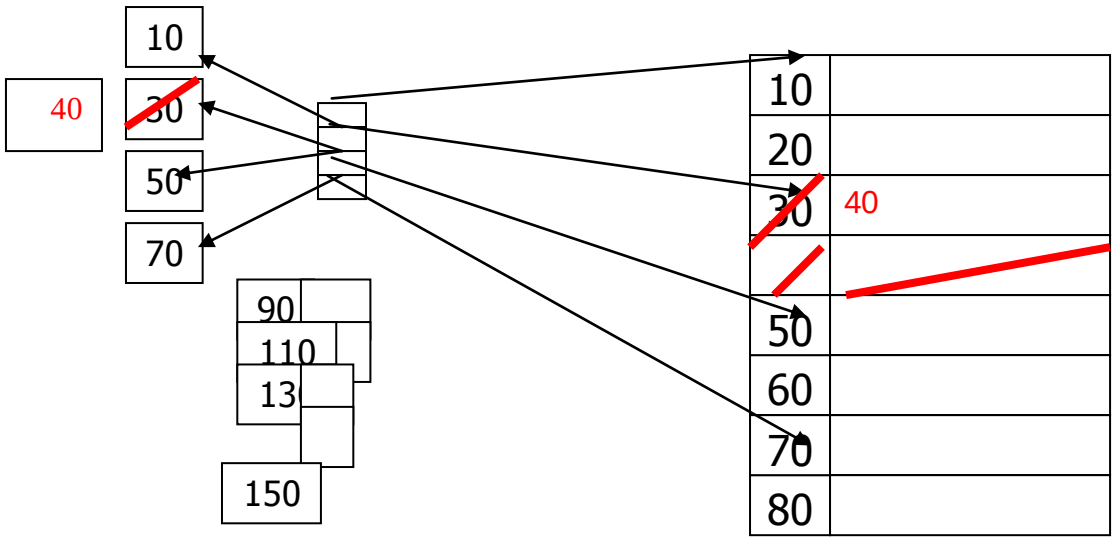
Törlés:

Tegyük fel hogy k1 kulcsú rekordot kívánjuk törölni.

Ehhez először keressük meg az a blokkot mely a rekordot tartalmazza, legyen ez Bi. Ha a k1 kulcs a blokkban nem a legkisebb, akkor a rekordot egyszerűen töröljük, a helyén keletkező űrt a rekordok blokkon belüli mozgatásával orvosolhatjuk.

Törlés ritka indexből II.

Töröljük a 30-as rekordot!



Módosítás:

Módosítás során két lehetőség áll fenn. Vagy egyszerű, ha a módosítás nem érinti a rekord kulcsát, ekkor csak megkeressük a módosítandó rekordot, elvégezzük a módosítást, majd visszaírjuk a háttértárba.
A bonyolultabb eset ha kulcsmezőt érint. Ekkor egy törlést követő beszúrással valósíthatja meg egy rekord módosítását.

B*- fák, mint többszintes ritka indexek:

Az indexelt szervezésnél log2N-nel arányos átlagos keresési idő érhető el, ami kisebb mint a heap szervezésé, de nagyobb mint a hash szervezésé, cserébe viszont a háttértár kihasználtsága változó méretű adatbázis esetén is kézben tartható.

Egy csomópontban ábrázolt k - mutatóhoz csak k-1 kulcsot tárolnak, mert a kulcs jelentése a kijelölt részfában tárolt legkisebb kulcsérték. Így az indexblokkok első kulcsérték bejegyzése nem hordozna információt. Az ilyen indexelést nevezik B*- fa indexnek.

Keresés:

Az eljárás hasonló a ritka indexeknél tanultakhoz, csupán az indexállományban keresést végezzük több lépésben.
Tételezzük fel, hogy a v1 kulcsú rekordra van szükségünk. Az indexállomány csúcsán álló blokkban megkeressük azt a rekordot, amelyiknek v2 kulcsa a legnagyobb azok közül, amelyek még kisebbek v1-nél. Ennek a rekordnak a mutatója az eggyel alacsonyabb szintű indexben rámutat arra a blokkra, amelyben a keresést tovább kell folytatni egy olyan indexrekord után, amelyiknek V3 kulcsa a legnagyobb azok közül, amelyek még kisebbek v1-nél. Az eljárás mindaddig folytatandó, ameddig az utolsó mutató már az adatállomány egy blokkját azonosítja, amelyben a kulcsú rekordnak lennie kell.

Beszúrás:

Annyiban tér el a ritka indexben összefoglaltaktól, hogy az indexállomány karbantartásánál gondosan ügyelni kell arra, hogy az eredeti fastruktúrát, annak kiegyenlítettségét fenntartsuk.

Törlés:

Megkeressük a keresett adatot és töröljük. Az adatblokkokat ha csak lehet összevonjuk. Összevonáskor, vagy ha egy adatblokk utolsó rekordját is töröljük, a megszűnt blokkhoz tartozó kulcsot is ki kell venni az index állomány érintett részfájából.

Módosítás:
Azonos a ritka indexnél tárgyaltakkal.

Sűrű indexek:

A ritka indexelés hátránya hogy az adatállományt rendezetten kell tárolni. Emiatt nincs mód arra, hogy egy-egy új rekordot tetszőleges szabad helyre szúrjunk be → csökken a háttértár kihasználtsága.
Megoldás lehet ha minden adatrekordhoz tartozik indexrekord. Az indexrekord továbbra is csak a rekordot tartalmazó blokkot azonosítja. Ezzel a megoldással a blokkon belüli keresési idő csökkenthető.
A sűrű indexek elsősorban a fő állomány kezelését könnyítik meg, illetve a több kulcs szerinti keresést teszik lehetővé.

Hátrányai:

- Plusz helyigény
- Eggyel több lapelérés kell egy rekord kiolvasásához
- Plusz adminisztrációval jár a karbantartása

Előnyei:

- Nincs szükség rendezett tárolásra → helymegtakarítás
- Felgyorsítja a rekordelérést
- Több kulcs szerinti keresés
- Az adatállomány rekordjai szabaddá tehetők, ha minden további rekordhívás a sűrű indexen keresztül történik.

Keresés:

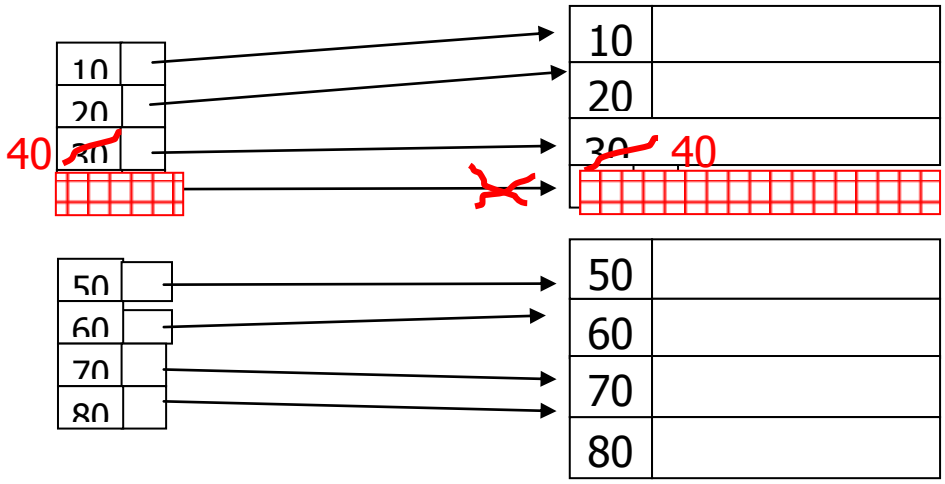
Az indexelt állományban megkeressük a kulcsot, a hozzá tartozó mutatóval elérhetjük a tárolt rekordot.

Törlés:

Megkeressük a rekordot, foglaltsági jelzését szabadra állítjuk, a kulcsot kivesszük az index állományból, és az indexállományt időnként tömörítjük.

Törlés sűrű indexből

Töröljük a 30-as rekordot!



Beszúrás:

Keresünk egy üres helyet a rekordnak, ha nem találunk, akkor az állomány végére vesszük fel. Beállítjuk a foglaltsági jelzést, és beírjuk az adatokat.

Módosítás:

Megkeressük a rekordot tartalmazó adatblokkot, majd módosítást követően visszaírjuk a háttértárra. Ha kulcsmezőt érintünk akkor az indexállományt újra rendezzük.

Megszorítások, integritási feltételek, triggerek

A megszorítások olyan előírások, korlátozások, amelyekkel megadhatjuk az adatbázis tartalmára vonatkozó kívánságainkat. Ha ezeket az adatbázisrendszerrel egyszer, egy helyen közöljük, az mindig gondoskodni fog betartásukról, ha a felhasználói rendszerbe építenénk be, akkor sok helyen kellene megadni, módosítani.
A megszorításokat az adatbázisrendszer minden olyan akció során ellenőrzi, amely eredményeként az adatbázis tartalma úgy változhat, hogy az a megszorítást már nem elégítené ki. A megszorítások megadásuktól kezdve érvényesülnek (nincs visszamenőleges hatásuk). Késleltetett ellenőrzés végrehajtás a DEFFERRED kulcsszóval.

Kulcsok

Az egyedek egyértelmű azonosítására alkalmas. Megszorításként azt jelenti, hogy ellenőrizze azt, hogy a relációban nem fordulhat elő két olyan sor, amelyben a kulcsattribútumok értéke páronként azonos lenne. Egy relációban több kulcs is lehetséges. Ezek közül szokás egy elsődleges kulcsot megadni. CREATE TABLE utasításnál megadható. Ha a kulcs egy attribútumos, akkor PRIMARY KEY vagy UNIQUE kulcsszavakkal adhatjuk meg, vagy az utasítás végén, ugyanezen kulcsszavak után zárójelben megadva felsorolásként. Ha nem egy attribútumos, akkor csak tábla végén.

A következő példában létrehozunk ugyanolyan megszorításokkal rendelkező táblákat hozunk létre,
a megszorítások két lehetséges megadási módját szemléltetve:

```
CREATE TABLE hallgatok(
neptun_kod CHAR(6) PRIMARY KEY,
nev VARCHAR2(30) NOT NULL,
szul_ev NUMBER(4) CHECK (szul_ev > 1900),
telefonszam NUMBER(10) UNIQUE);
CREATE TABLE atlagok(
neptun_kod CHAR(6) REFERENCES hallgatok(neptun_kod),
atlag NUMBER(3,2));
```

```
CREATE TABLE hallgatok2(
neptun_kod CHAR(6),
```

```
nev VARCHAR2(30) NOT NULL,
szul_ev NUMBER(4),
telefonszam NUMBER(10),
PRIMARY KEY (neptun_kod),
CHECK (szul_ev > 1900),
UNIQUE (telefonszam)
);
```

```
CREATE TABLE atlagok2(
neptun_kod CHAR(6),
atlag NUMBER(3,2),
FOREIGN KEY (neptun_kod) REFERENCES hallgatok2(neptun_kod)
)
```

A fenti attribútumok tehát a következő megszorításokkal rendelkeznek:

• a hallgatok és hallgatok2 táblában a neptun_kod elsődleges kulcs, tehát minden sorban különböző értéknek kell szerepelnie és nem lehet köztük null;

• a név mező nem tartalmazhat null értéket;

• a születési évnek nagyobbnak kell lennie, mint 1900;

• nem szerepelhet két azonos telefonszám, de előfordulhat, hogy valakinek nem ismerjük a telefonszámát;

• az átlagok és átlagok2 táblákba csak olyan rekord kerülhet, melyben a neptun_kod mező értéke

szerepel már a hallgatok tábla neptun_kod attribútumának értékei között.

Lássunk egy példát arra, hogyan adhatunk még attribútum halmazt elsődleges kulcsként (UNIQUE és

FOREIGN KEY hasonlóan adható meg attribútum halmazra):

```
CREATE TABLE hallgatok(
nev VARCHAR2(30),
szul_datum DATE,
anyja_neve VARCHAR2(30),
lakcim VARCHAR2(100),
PRIMARY KEY(nev, szul_datum, anyja_neve));
```

Ebben a táblában a nev, szul_datum és anyja_neve attribútumok vehetnek fel azonos értékeket két

rekordban külön-külön, de hogy mindhárom attribútum megegyezzen, arra minimális az esély, ezért

használják a gyakorlatban gyakran elsődleges kulcsként.

Hivatkozási épség megszorítás

~ azt jelenti, hogy egy reláció bizonyos attribútumaiban szereplő értékek csak olyan értékek lehetnek, amelyek egy adott másik táblában előforduló elsődleges kulcsértékek (idegen kulcsok). A hivatkozási épség előírás megadására két lehetőség van. Ha az idegen kulcs egy attribútumos, akkor az attribútum definiálásakor: REFERENCES tábla (attribútum) megadásával vagy a CREATE TABLE utasítás végén: FOREIGN KEY attribútumok REFERENCES tábla (attribútumok), módon. Ha az idegen kulcs nem egy attribútumos, akkor csak az utóbb megadási lehetőség van.

A hivatkozási épség sérülhet oly módon, hogy a hivatkozó táblába módosítás vagy beszúrás során olyan érték kerülne, amely a hivatkozott táblában nem fordul elő a megnevezett attribútumokon, és sérülhet úgy is, hogy a hivatkozott táblában módosítunk, vagy törlünk olyan sorokat, amelyek korábban helyes hivatkozások ottani párjai voltak. Az adatbázisrendszerek a hivatkozási épség megsértésekor nemcsak az ezt sértő beavatkozás tiltásával élhetnek, hanem kétféle reagálás lehetőségét kínálják fel:

- A módosítások visszautasítása
- Tovagyűrűző eljárás (CASCADE). Ha a hivatkozott helyen módosítjuk a hivatkozásban résztvevő attribútum értékeket, akkor a hivatkozó helyeken is ennek megfelelően automatikusan módosítja az adatbázisrendszer a hivatkozó értékeket, ezzel helyreállítja a hivatkozást.
- NULL értékre állítás módszere, (SET NULL). Ha a hivatkozott helyen történő változás miatt a hivatkozás sérülne, akkor a hivatkozó helyen a hivatkozó értéket NULL-ra állítja.

Azt, hogy sérülés esetén ezek közül mi legyen az adatbázis reakciója azt a hivatkozó táblát létrehozó CREATE TABLE utasításban, a hivatkozás megadásával együtt adhatjuk meg. ON DELETE SET NULL: törlés miatti sérülés esetén a hivatkozó érték NULL-ra állítódik.

ON UPDATE CASCADE módosítás esetén, a hivatkozás helyén is történjen meg az attribútum értékek módosítása, hogy a hivatkozás ismét érvényes legyen.

Hivatkozási épség megszorítások: Előírhatjuk, hogy 1 attribútum v. attribútum-halmaz értékeinek elő kell fordulnia 1 másik reláció valamelyik sorának elsődleges kulcs attribútuma(i)ban. Ezt a relációséma megadásakor a REFERENCES v. a FOREIGN KEY kulcsszóval adhatjuk meg:

- Ha az idegen kulcs egyetlen attribútum:

```
REFERENCES <tábla> (<attribútum>)
```

- Ha az idegen kulcs több attribútumból áll:

```
FOREIGN KEY <attribútumok> REFERENCES <tábla>  (<attribútumok>)
```

Az idegen kulcsot kétféleképpen deklarálhatjuk, ugyanúgy ahogyan azt az elsődleges kulcsok esetében tettük.

Attribútum értékekre vonatkozó megszorítások

Az attribútumok lehetséges értékeik a típusuk megadása még alig korlátozza. Hibás adat bevitelének lehetőségek, hibásra módosítás lehetőségét csökkentik.
NOT NULL feltétel. Megadásával előírhatjuk, hogy az adott attribútumnak mindig érvényes értékkel kell rendelkeznie, értéke sosem lehet NULL. Megadása a relációt definiáló CREATE TABLE utasításban az érintett attribútum definiálásakor.
CHECK feltétel. Megadásával olyan korlátozásokat előírhatunk, mint a WHERE után. Megengedett értékek, aritmetikai kifejezések. Megadása a relációt definiáló CREATE TABLE utasításban az érintett attribútum definiálásakor.
A CHECK feltétellel nemcsak attribútum értékekre vonatkozó kikötéseket adhatunk, hanem sorokra vonatkozó megszorítást is. Ilyenkor a CHECK feltételt nem egy attribútum definiálásakor fűzzük, hanem a táblát definiáló utasítás végén adjuk meg.

Egy tábla definiálásakor a tábla nevén és az attribútumokon kívül egyéb információt is lehet megadni. Ilyenek a

kulcsok
az attribútum értékekre vonatkozó megszorítások.

Először a kulcsok, és az egyedi értékekkel bíró attribútumok megadásának módját ismertetjük. Az SQL-ben alapvetően az *elsődleges kulcs* megadására van lehetőségünk, ahogy azt a legtöbb ABKR megköveteli. Ha az elsődleges kulcsot szeretnénk definiálni, akkor a tábla létrehozását kibővíthetjük megfelelő opciókkal (záradékokkal). Ez a következőképpen néz ki:

```
CREATE TABLE <táblanév> {
<attribútumdefiníció> [UNIQUE]  [,<attribútumdefiníció> [UNIQUE]]…
[,PRIMARY KEY (<kulcsattribútum>
[,<kulcsattribútum>]…)|UNIQUE(<kulcsattribútum>) ]}
```

A **UNIQUE** kulcsszó segítségével minden egyes attribútumnál megadhatjuk, hogy az adott attribútum csak egyedi értékeket vehet fel. A <*kulcsattribútum*> paraméterben kell megadni annak az attribútumnak a nevét, amely a kulcsot alkotja, vagy annak egy részét képezi. Amennyiben csak egy attribútum tartozik a kulcshoz, akkor használhatjuk mind a **PRIMARY KEY,** mind a **UNIQUE**parancsokat. Több attribútumból álló kulcsot csak a **PRIMARY KEY** kulcsszóval definiálhatunk.

Önálló megszorítások

Az SQL2 lehetőséget ad olyan megszorítások megadására, amely bármilyen feltétel ellenőrzését lehetővé teszi.
Általános alakja: CREATE ASSERTION megszorításnév CHECK feltétel;
Az önálló megszorítás lehetőségeit az SQL3-ban bővítették. Az ellenőrzést a programozó által megadott események váltják ki, valamint a megszorítás a tábla/táblák egyes soraira is vonatkozhat, nemcsak a teljes táblákra.

Megszorítások módosítása

Azért, hogy a megszorítást módosítani, törölni tudjuk, a létrehozásakor nevet kell neki adni. A névadást definiáláskor lehet megtenni. A megszorítás nevét a CONSTRAINT kulcsszó után adhatjuk meg. Például: CONSTRAINT titulus CHECK (paraméterek).
Ha a névvel megnevezett megszorítást törölni kívánjuk, akkor azt az ALTER TABLE táblanév DROP CONSTRAINT megszorításnév;
utasítással tehetjük meg.
Új megszorítás: ALTER TABLE táblanév ADD CONSTRAINT megszorításnév megszorítás definiálása;

Tábla szintű megszorítás:

```
ALTER TABLE <tábla név> ADD CONSTRAINT <név> <típus> <oszlop>;
```

Példafeladat:

Adjunk egy olyan megszorítást a tTanar táblához, aminek következtében nem tárolhatunk két azonos nevű tanárt.

```
ALTER TABLE tTanar ADD CONSTRAINT uq_tTanar UNIQUE  (Nev);
```

Ellenörzés:

```
INSERT INTO tTanar  VALUES (1, ' Példa Béla');
```

Megszorítás Törlése:

```
ALTER TABLE <tábla név> DROP CONSTRAINT <megszorítás név>;
```

Példa:

Dobjuk el az előbbi megszorítást:

```
ALTER TABLE tTanar DROP CONSTRAINT  uq_tTanar;
```

Ellenőrzés:

```
INSERT INTO tTanar VALUES  (1, ' Példa Béla');
```

Az adatbázis konzisztencia megőrzése.

Konzisztencia sorozatok

Triggerek: (Oracle 10g)

A trigger olyan tevékenységet definiál mely automatikusan végbemegy, ha egy tábla vagy nézet módosul vagy ha egyéb felhasználói vagy rendszeresemények következnek be. Azaz bármilyen változás az adatbázisban egy triggert indít el. A trigger egy adatbázis- objektum.
A triggerek működése felhasználói szemszögből átlátszóan működik.

Működését kiválthatja:

- Egy táblán vagy nézeten végrehajtott INSERT, DELETE vagy UPDATE utasítás
- Egyes DDL utasítások
- Szerverhibák
- Felhasználói ki és bejelentkezés
- Adatbázis elindítása leállítása

Az alábbi esetekben használjuk őket:

- származtatott oszlopérték generálása
- érvénytelen tranzakció megelőzése
- védelem
- hivatkozási integritási megszorítások definiálása
- komplex üzleti szabályok kezelése

- eseménynaplózás
- követés
- táblastatisztikák gyűjtése
- adattöbbszörözés

A tiggereket többféle szempont alapján osztályozhatjuk. Egy triggernél meg kell határozni, hogy az eseményhez viszonyítva mikor és hányszor fusson le. Ez alapján a **következő triggerekről beszélhetünk:**

- sor és utasításszintű trigger
- Before és After trigger
- Instead Of trigger
- Rendszer triggerek

Sor szintű trigger:

Röviden összefoglalva annyiszor fut le ahányszor a tábla adatai módosulnak. PL: Egy Delete utasításnál minden törölt sor aktiválja a triggert. Azonban ha nem módosul egyetlen sor sem akkor nem fut le egyszer sem a trigger.

A trigger megszámolja a 100 000-nél kisebb fizetésű új dolgozókat:

```
CREATE TRIGGER dolg_szamlal
AFTER INSERT ON dolgozo
FOR EACH ROW
WHEN (NEW.fizetes < 100000)
BEGIN
UPDATE szamlalo SET ertek=ertek+1;
END;
```

Utasítás szintű trigger:

Ez a trigger ellentétben a sor szintű társával csak egyetlen egyszer fut le, és akkor is lefut ha nem történt változás az adatbázisban.

Egyelemű segédtábla:

```
CREATE TABLE szamlalo (ertek NUMBER);
INSERT INTO szamlalo VALUES (-1);
```

Több új dolgozó felvételének esetére két triggert definiálunk. Az első nullázza a számlálót:

```
CREATE TRIGGER dolg_kezdo
BEFORE INSERT ON dolgozo
BEGIN
UPDATE szamlalo SET ertek=0;
END;
```

Before és After triggerek:

Egyaránt lehetnek sor és utasítás szintűek. Csak táblához kapcsolhatók nézethez nem, ám egy alaptáblához kapcsolt trigger lefut a nézeten végrehajtott DML utasítás esetén is.

A Before trigger a hozzákapcsolt utasítás előtt fut le, míg az After trigger nevéből adódóan az utasítás után fut le.

Before:

```
CREATE OR REPLACE TRIGGER emp_alert_trig
BEFORE INSERT ON emp
BEGIN
DBMS_OUTPUT.PUT_LINE('New employees are about to be
added');
END;
```

Szúrjunk be egy sort!

```
INSERT INTO emp(empno, ename, deptno) VALUES(8000, 'valaki',
40);
```

After:

Hozzuk létre az új táblát, ahol a módosításokat fogjuk letárolni!

```
CREATE TABLE empauditlog (
audit_date DATE,
audit_user VARCHAR2(20),
audit_desc VARCHAR2(20)
);
```

Hozzuk létre a triggert!

```
CREATE OR REPLACE TRIGGER emp_audit_trig
AFTER INSERT OR UPDATE OR DELETE ON emp
DECLARE
v_action VARCHAR2(20);
BEGIN
IF INSERTING THEN
v_action := 'Added employee(s)';
ELSIF UPDATING THEN
```

```
v_action := 'Updated employee(s)';
ELSIF DELETING THEN
v_action := 'Deleted employee(s)';
END IF;
INSERT INTO empauditlog VALUES (SYSDATE, USER, v_action);
END;
```

Instead of trigger:

Ez a trigger a hozzákapcsolt utasítás helyett fut le. Csak sor szintű lehet és csak nézeteken definiálható. Ha egy nézetet módosítani akarunk, de az nem tehetjük közvetlenül a DML utasítások segítségével, akkor használjuk az Instead Of triggert.

Rendszer triggerek:

Célja az előfizetők tájékoztatása az adatbázis eseményekről.

Már tudjuk hogy mik a triggerek, mikor lépnek működésbe, mikor használjuk őket, és milyen fajtái vannak. Azonban még azt nem tudjuk, hogy hogyan hozhatóak létre. A következő részben ezt fogjuk megvizsgálni.

Triggerek létrehozása:

Saját sémában ----- CREATE TRIGGER

Másik felhasználó sémájában ----- CREATE ANY TRIGGER

Az adatbázisban létrehozandóhoz pedig ----- ADMINISTER DATABASE TRIGGER

Jogosultság szükséges.

Létrehozó utasítás:

```
CREATE [OR REPLACE] TRIGGER [séma. ] triggernév
{ BEFORE | AFTER | INSTEAD OF }
{dml_trigger | { ddl_esemény  [OR ddl_esemény] …|
Ab_esemény [OR  ab_esemény]…}
ON {DATABASE | [séma. ] SCHEMA}
[WHEN (feltétel) ]   {plsql_blokk | eljáráshívás}
```

Ahol

Dml_trigger::=

```
{INSERT | DELETE | UPDATE  [OF oszlop [ , oszlop]…}
[OR {INSERT | DELETE | UPDATE  [OF oszlop [ , oszlop]…} ]….
ON { [ séma. ] tábla |
[ NESTED TABLE  bát_oszlop OF ]  [séma. ] nézet}
[REFERENCING {OLD  [AS] régi | NEW [AS] új | PARENT [AS] szülő}
[ { OLD [AS]  régi | NEW [AS] új | PARENT [AS] szülő} ]…
[FOR EACH ROW]
```

Vegyük sorba mit miért írtunk:

Az OR REPLACE a már létező trigger újradefiniálása, annak előzetes megszüntetése nélkül.
A séma triggert tartalmazó séma neve. Ha hiányzik, akkor a parancsot kiadó felhasználó sémájában jön létre a trigger, egyszóval nem ott ahol azt mi szeretnénk.
A trigger név a létrehozandó trigger neve.
A BEFORE, AFTER, INSTEAD OF a trigger típusát adja meg.
Az INSERT, DELETE, UPDATE definiálja azt az SQL utasítást, melynek hatására a trigger lefut.
Az ON utasításrész azt az adatbázis-objektumot adja meg, melyen a triggert létrehozzuk.
A REFERENCING utasításrész korrelációs neveket(régi, új, szülő) határoz meg.
A NEW a módosítás utáni neveket adja meg.
A FOR EACH ROW sor szintű triggereket hoz létre.

A ddl_esemény egy olyan DDL utasítást, az ab_esemény egy olyan adatbázis-eseményt határoz meg, amelyek a triggert aktiválják.

Már csak egyetlen kérdés maradt a triggerekkel kapcsolatban: Hogyan működnek?

A triggerek Működése:

Egy triggernek két állapota lehet: engedélyezett és letiltott. A letiltott trigger nem indul el, ha a kiváltó esemény bekövetkezik. Az engedélyezett trigger esetén az Oracle automatikusan a következő eseményeket hajtja végre:

- lefuttatja a triggert, ha ugyanarra az utasításra több azonos típusú trigger van definiálva, akkor sorrendjük nincs meghatározva.
- Ellenőrzi a megszorításokat és felügyeli a triggereket hogy ne sértsék meg azokat.
- Olvasási konzisztenciát biztosít a lekérdezésekhez.
- Kezeli a trigger és a sémaobjektumok közötti függőségeket.
- Osztott adatbázis esetén, ha a trigger távoli táblát módosított, kétfázisú véglegesítést alkalmaz.

A CREATE utasítás automatikusan engedélyezi a triggert.

Triggert letiltani és engedélyezni az ALTER TRIGGER és az ALTER TABLE utasítással lehet.

Azt már tudjuk hogy ha ugyan azon utasításon azonos triggerek vannak definiálva akkor nincs sorrend. De mi a helyzet ha nem azonosak?

Az Oracle egy végrehajtási modellt követ:

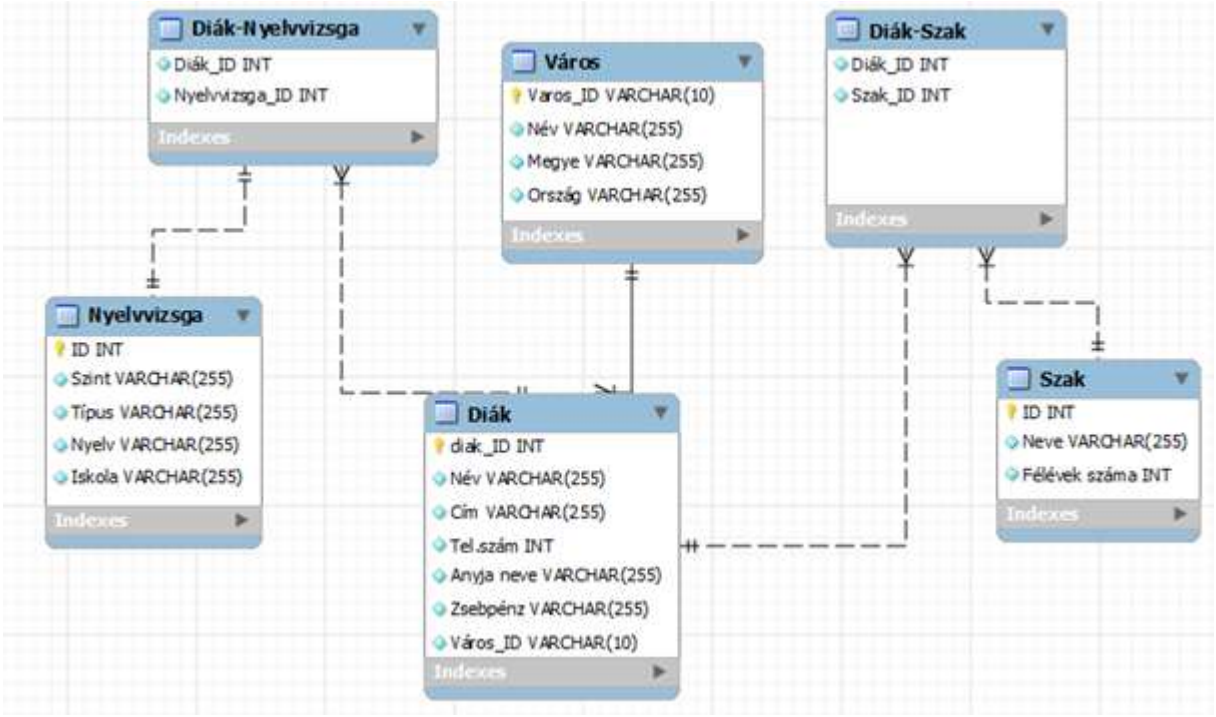
- Végrehajtja az összes utasításszintű BEFORE triggert.
- A DML utasítás által érintett minden sorra ciklikusan:
  a, végrehajtja a sorszintű triggereket
  b, zárolja és megváltoztatja a a sort és ellenörz az integritási megszorításokat.
  a zár csak a tranzakció végeztével oldódik,
  c, végrehajtja a sorszintű AFTER triggereket.
- Ellenörzi a késleltetett integritási megszirításokat.
- Végrehajtja az utasítás szintű AFTER triggereket.

A végrehajtási modell rekurzív. Egy trigger működése közben újabb triggerek indulhatnak el.

Feladatok

1. feladat

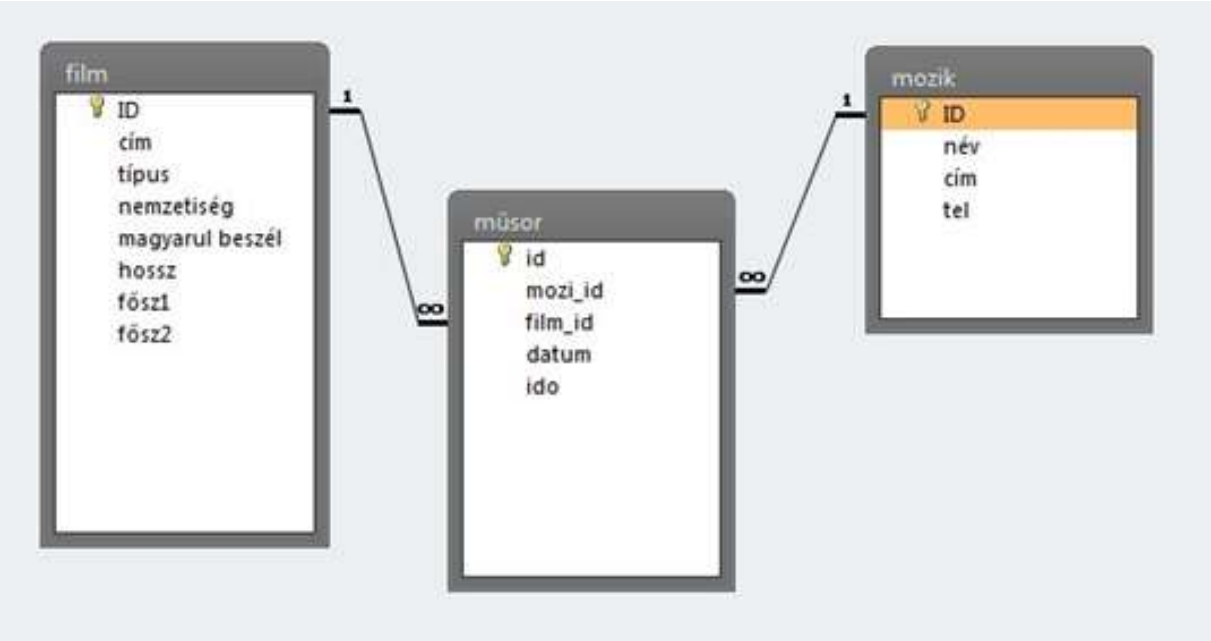
Adott egy adatbázis az alábbi szerkezettel.



1. Hozza létre az adatbázis táblákat, a saját adatbázisában.
2. Töltsön a táblákba 5-5 megfelelő rekordot.
3. Listázza ki a diákok nevét, zsebpénzét név szerint csökkenő sorrendben.
4. Listázza ki a szakok nevét, az adott szakon tanuló diákok nevével. Rendezzen szak neve szerint és azon belül diák neve szerint.
5. Számolja össze, hogy megyénként hány diák tanul az iskolában.
6. Kik azok a diákok, akiknek az átlagosnál kevesebb a zsebpénze? Listázza ki a nevüket, és a zsebpénzüket, zsebpénz szerint csökkenő sorrendben.
7. Írjon tárolt eljárást, mely meghatározza a legnagyobb különbséget a zsebpénzek között, és ezzel az értékkel csökkenti a legnagyobb zsebpénz összegét.
8. Írjon tárolt eljárást, mely az Egerben lakó diákok zsebpénzét megnöveli 10%-al ha az átlagos zsebpénznél magasabbak, és 15%-al, ha alacsonyabbak.

2. feladat

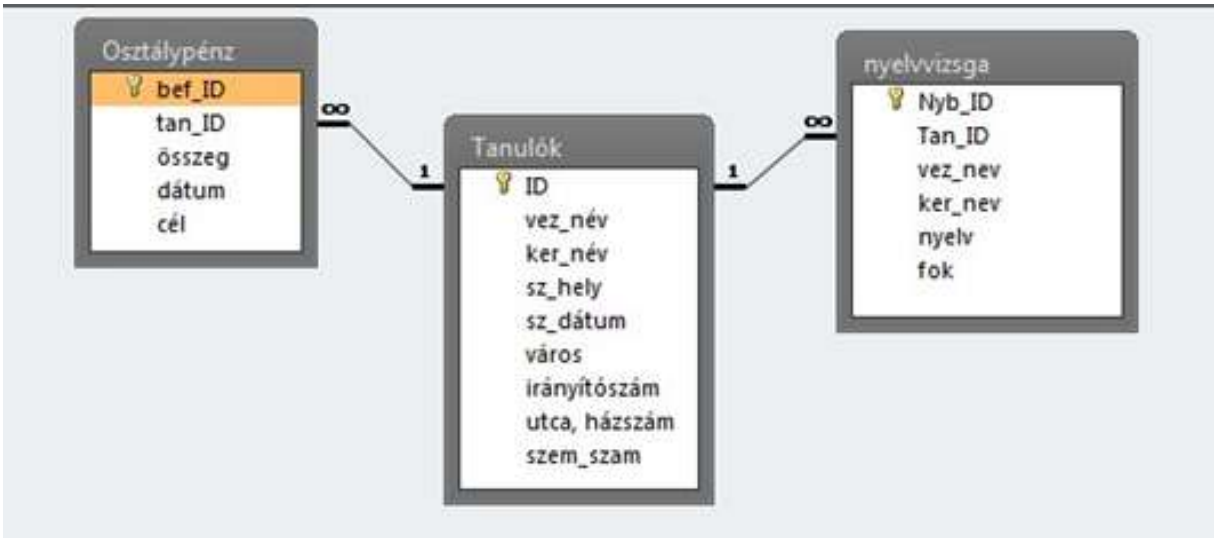
Adott egy adatbázis az alábbi szerkezettel.



1. Írd le a mozik táblát létrehozó utasítást!
2. Vigyél fel egy új mozit a táblába. Az adatai: neve: Csillag, címe: Eger, Leányka út 4, 3300, és a telefonszámot nem tudjuk. Az elsődleges kulcs identity típusú.
3. Módosítsd a Penge című film típusát romantikusra és hosszát 135 percre.
4. Törölje a vasárnapi (április 22) műsor minden filmjét áramszünet miatt.
5. Készíts műsor-listát az áprilisi filmekről !
6. Hány darab filmet vetítettek márciusban az Uránia moziban?
7. Listázd ki azokat a filmeket, melyek hossza rövidebb, mint az átlagos filmhossz.
8. Melyik moziban vetítik a legtöbb filmet?
9. Hány darab filmet találunk az adatbázisban nemzetiségenként? Rendezd ezeket csökkenő sorrendben!
10. Hány moziban vetítik 2012-ben a Harry Potter 7 második részét?
11. Írj tárolt eljárást, mely az összes 120 percnél hosszabb film percben mért hosszát összeadja.

3. feladat

Adott egy adatbázis az alábbi szerkezettel.



1. Írd le a Tanulók táblát létrehozó utasítást.
2. Új tanuló érkezett az osztályba, akinek az adatait be kell vezetni az osztály tanulóinak adatait tartalmazó adatbázisba (osztály.mdb, tanulók tábla). A tanuló adatai a következők: név: Kovács Péter, születési hely: Budapest, születési dátum: 1992. jún. 12., cím: 2120 Dunakeszi, Munkácsy u. 12. (Az azonosító a vezetéknév és a keresztnév kezdőbetűiből és egy számból áll, mely szám azt jelöli, hogy hányadik tanulónak van ilyen monogramja. Itt: KP1)
3. Vágó László (VL1) születési helyét rosszul írta be az osztályfőnök, Bécs helyett Pécsen született. Javítsd ki a születési helyet az osztály adatbázis tanulók táblájában!
4. Kovács Péter (KP1) szülei úgy döntöttek, hogy mégis más iskolába viszik a gyereküket, így Kovács Pétert ki kell törölni az osztály tanulóinak névsorából.
5. Szűrd ki a tanulók táblából azokat, akik B-vel vagy V-vel kezdődő helyen születtek és rendezd őket születési dátum szerint növekvő sorrendbe.
6. Számold össze, hogy az egyes nyelvekből külön-külön hány tanulónak van nyelvvizsgája.
7. Készítsd el tanulónként az összes befizetést tartalmazó listát. Rendezd névsorba, azon belül összes befizetés szerint csökkenőleg.
8. Hány tanuló befizetése van az átlag befizetés felett?
9. Melyik városból jött az iskolába a legtöbb tanuló?
10. Mely tanulók nem fizettek még be osztálypénzt?
11. Írj tárolt eljárást, mely kiválogatja az angol és a német középfokú nyelvvizsgával rendelkező tanulókat, és összeveti az általuk befizetett osztálypénz átlagát.

A PL/SQL alapjai

A PL/SQL alapvető elemei

Karakterkészlet

Egy PL/SQL program forrásszövegének (mint minden forrásszövegnek) a legkisebb alkotóelemei a karakterek. A PL/SQL nyelv karakterkészletének elemei a következők:

• betűk, az angol kis- ás nagybetűk: A—Z és a—z;
számjegyek: 0—9;
• egyébkarakterek: ()+-/<>=!~^;:.@%,#$&{}?[]
• szóköz, tabulátor, kocsi vissza.
A PL/SQL-ben a kis- és nagybetűk nem különböznek, a sztring literálok belsejét kivéve.

Lexikális egységek

A PL/SQL program szövegében a következő lexikális egységek használhatók:

- elhatárolók,
- szimbolikus nevek,
- megjegyzések,
- literálok.

Ezeket részletezzük a következőkben.

Elhatárolók

Szimbólum	Jelentés
+	Összeadás operátora
%	Attribútum kezdőszimbóluma
'	Sztring literál határoló
.	Komponens szelektor
/	Osztás operátora
(	Részkifejezés vagy lista kezdete
)	Részkifejezés vagy lista vége
:	Gazdaváltozó kezdőszimbóluma
,	Felsorolás elválasztójele

*	Szorzás operátora
"	Idézőjeles azonosító határolójele
=	Hasonlító operátor
<	Hasonlító operátor
>	Hasonlító operátor
@	Távoli hivatkozás szimbóluma
;	Utasítás végjele
-	Kivonás és negatív előjel operátora

Szimbólikus nevek

Azonosítók: betűvel kezdődik és betűvel, számjeggyel vagy $ _ # karakterekkel folytatódik. Nagy és kisbetűk nem különböznek!

hallgato, HALGGATO, Hallgato
ugyanazt jelenti

További szimbólumok:

Szimbólum	**Jelentés**
:=	Értékadás operátora
=>	Hozzárendelés operátora
||	Összefűzés operátora
**	Hatványozás operátora
<<	Címke kezdete
>>	Címke vége
/*	Megjegyzés kezdete
*/	Megjegyzés vége
..	Intervallum operátora
<>	Hasonlító operátor
!=	Hasonlító operátor
~=	Hasonlító operátor
^=	Hasonlító operátor
<=	Hasonlító operátor
>=	Hasonlító operátor
--	Egysoros megjegyzés kezdete

Foglalt szavak

(mod, or and, declare, …)

Idézőjeles azonosítók:

„Igen/Nem”

„mod”

„Ez is azonosító”

Az idézőjeles azonosítók karakterhelyesen tárolódnak!

Literálok

5, 33, 6.666, -1.0, 2.0f (BINARY_FLOAT),

2.1d (BINARY_DOUBLE)

’almafa’, ’mondta’

Címke

<< címke >>

Bármely sorban lehet. Azonosító!

Nevesített konstans

```
név CONSTANT típus [NOT NULL] {:=|DEFAULT} kifejezés;
```

VÁLTOZÓ

```
név típus [NOT NULL] {:=|DEFAULT} kifejezés;
```

Például:

```
DECLARE
  -- NOT NULL deklarációnál kötelező az értékadás
  v_Szam1 NUMBER NOT NULL := 10;

  -- v_Szam2 kezdőértéke NULL lesz
  v_Szam2 NUMBER;

BEGIN
  v_Szam1 := v_Szam2; -- VALUE_ERROR kivételt eredményez
END;
```

```
-- nevesített konstans deklarációja, az értékadó kifejezés
függvényhívás
  c_Most       CONSTANT DATE := SYSDATE;
  -- változódeklaráció kezdőértékadás nélkül
```

```
  v_Egeszszam  PLS_INTEGER;
  v_Logikai    BOOLEAN;
  -- változódeklaráció kezdőértékadással
  v_Pozitiv    POSITIVEN DEFAULT 1;
  v_Idopecset  TIMESTAMP := CURRENT_TIMESTAMP;
  -- kezdőértékadás rekordtípus mezőjének
  TYPE t_Kiserlet IS RECORD (
     leiras       VARCHAR2(20),
     probalkozas  NUMBER := 0,
     sikeres      NUMBER := 0    );
  -- rekord mezőinek csak a típus deklarációban lehet kezdőértéket
adni
  v_Kiserlet   t_Kiserlet;
```

Egyszerű és összetett típusok

A literálok, nevesített konstansok, változók mindegyike rendelkezik adattípus komponenssel, amely meghatározza az általuk felvehető értékek tartományát, az értékeken végezhető műveleteket és az értékek tárbeli megjelenítési formáját.

Az adattípusok lehetnek rendszerben definiált vagy felhasználói típusok.

Skalártípusok:

SKALÁRTÍPUSOK		
Numerikus család	Karakteres család	Dátum/intervallum család
BINARY_DOUBLE	CHAR	DATE
BINARY_FLOAT	CHARACTER	INTERVAL DAY TO SECOND
BINARY_INTEGER	LONG	INTERVAL YEAR TO MONTH
DEC	LONG RAW	TIMESTAMP
DECIMAL	NCHAR	TIMESTAMP WITH TIME
DOUBLE PRECISION	NVARCHAR2	ZONE

FLOAT	RAW	TIMESTAMP WITH LOCAL
INT	ROWID	TIME ZONE
INTEGER	STRING	
NATURAL	UROWID	
NATURALN	VARCHAR	
NUMBER	VARCHAR2	
NUMERIC		
PLS_INTEGER		
POSITIVE		
POSITIVEN	Logikai család	
REAL	BOOLEAN	
SIGNTYPE		
SMALLINT		
ÖSSZETETT TÍPUSOK	LOB TÍPUSOK	REFERENCIATÍPUSOK
RECORD	BFILE REF	CURSOR
TABLE	BLOB	SYS_REFCURSOR
VARRAY	CLOB	REF objektumtípus
NCLOB		

NUMBER TÍPUS

Ez a típus egész és valós számokat egyaránt képes kezelni. Megegyezik a NUMBER adatbázistípussal. Tartománya: 1E–130..10E125. Belső ábrázolása fixpontos vagy lebegőpontos decimális. Szintaxisa a következő:

```
NUMBER [(p[,s])]
```

A teljes alakban p a pontosság, s a skála megadását szolgálja. Értékük csak egész literál lehet. A pontosság az összes számjegy számát, a skála a tizedes jegyek számát határozza meg.

Típus	Kezelendő érték	Tárolt érték
NUMBER	123.456	123.456
NUMBER(3)	321	321

NUMBER(3)	3210	Túlcsordulás
NUMBER(4,3)	11.2222	Túlcsordulás
NUMBER(4,3)	3.17892	3.1799
NUMBER(3,–3)	1234	1000
NUMBER(3,–1)	1234	1230

A NUMBER típus belső ábrázolása hatékony tárolást tesz lehetővé, de az aritmetikai műveletek közvetlenül nem végezhetők el rajta.
Ha egy egész értéket nem akarunk tárolni, csak műveletet akarunk vele végezni, akkor használjuk a BINARY_INTEGER adattípust. Ez a típus tehát egész értékeket kezel a –2147483647..2147483647 tartományban. Ezeket az értékeket fixpontosan tárolja, így a műveletvégzés gyorsabb.
A BINARY_INTEGER korlátozott altípusai a következők:

NATURAL	0..2147483647
NATURALN	0..2147483647 és NOT NULL
POSITIVE	1..2147483647
POSITIVEN	1..2147483647 és NOT NULL
SIGNTYPE	–1,0,1

Karakteres család

A karakteres típusok tartományának elemei tetszőleges karaktersorozatok. Reprezentációjuk
CHAR TÍPUS
Fix hosszúságú karakterláncok kezelésére alkalmas. Szintaxisa:

```
CHAR [(h [CHAR|BYTE])]
```

ahol h az 1..32767 intervallumba eső egész literál, alapértelmezése 1. A h értéke bájtokban (BYTE) vagy karakterekben (CHAR – ez az alapértelmezés) értendő és a hosszat adja meg. A karakterláncok számára mindig ennyi bájt foglalódik le, ha a tárolandó karakterlánc ennél rövidebb, akkor jobbról kiegészül szóközökkel.
VARCHAR2 TÍPUS

Változó hosszúságú karakterláncok kezelésére alkalmas. Szintaxisa a következő:

```
VARCHAR2(h [CHAR|BYTE])
```

ahol h az 1..32767 intervallumba eső egész literál és a maximális hosszat adja meg. Értéke CHAR megadása esetén karakterekben, BYTE esetén bájtokban, ezek hiánya esetén karakterekben értendő. A maximális hossz legfeljebb 32767 bájt lehet.
A megadott maximális hosszon belül a kezelendő karakterláncok csak a szükséges mennyiségű bájtot foglalják el.

ROWID, UROWID TÍPUSOK

Minden adatbázistábla rendelkezik egy ROWID nevű pszeudooszloppal, amely egy bináris értéket, a sorazonosítót tárolja. Minden sorazonosító a sor tárolási címén alapul. A fizikai sorazonosító egy „normális" tábla sorát azonosítja, a logikai sorazonosító pedig egy asszociatív tömbét. A ROWID adattípus tartományában fizikai sorazonosítók vannak. Az UROWID adattípus viszont fizikai, logikai és idegen (nem Oracle) sorazonosítókat egyaránt tud kezelni.

Dátum/intervallum típusok

Ezen családon belül három alaptípus létezik: DATE, TIMESTAMP és INTERVAL.

DATE TÍPUS

Ez a típus a dátum és idő információinak kezelését teszi lehetővé. Minden értéke 7 bájton tárolódik, amelyek rendre az évszázad, év, hónap, nap, óra, perc, másodperc adatait tartalmazzák.
A típus tartományába az időszámítás előtti 4712. január 1. és időszámítás szerinti 9999. december 31. közötti dátumok tartoznak. A Julianus naptár alkalmazásával az időszámítás előtti 4712. január 1-jétől eltelt napok számát tartalmazza a dátum.
Az aktuális dátum és idő lekérdezhető a SYSDATE függvény visszatérési értékeként.

TIMESTAMP TÍPUS

Ezen típus tartományának értékei az évet, hónapot, napot, órát, percet, másodpercet és a másodperc törtrészét tartalmazzák. Időbélyeg kezelésére alkalmas. Szintaxisa:

```
TIMESTAMP[(p)] [WITH [LOCAL] TIME ZONE]
```

ahol p a másodperc törtrészének kifejezésére használt számjegyek száma. Alapértelmezésben 6. A WITH TIME ZONE megadása esetén az értékek még a felhasználó időzónájának adatát is tartalmazzák. A LOCAL megadása esetén pedig az adatbázisban tárolt (és nem a felhasználói) időzóna adata kerül kezelésre.

INTERVAL TÍPUS

A INTERVAL típus segítségével két időbélyeg közötti időtartam értékeit kezelhetjük. Szintaxisa:

```
INTERVAL {YEAR[(p)] TO MONTH|DAY[(np)] TO SECOND[(mp)]}
```

A YEAR[(p)] TO MONTH az időintervallumot években és hónapokban adja. A p az évek értékének tárolására használt számjegyek száma. Alapértelmezett értéke 2.

A DAY[(np)] TO SECOND [(mp)] az időintervallumot napokban és másodpercekben adja; np a napok, mp a másodpercek értékének tárolására használt számjegyek száma; np alapértelmezett értéke 2, mp-é 6.

Logikai típus

Egyetlen típus tartozik ide, a BOOLEAN, amelynek tartománya a logikai igaz, hamis és a NULL értéket tartalmazza. Logikai típusú literál nincs, de az Oracle három beépített nevesített konstanst értelmez. TRUE értéke a logikai igaz, FALSE értéke a logikai hamis és NULL értéke NULL.

A rekordtípus

A rekord logikailag egybetartozó adatok heterogén csoportja, ahol minden adatot egy-egy mező tárol. A mezőnek saját neve és típusa van. A rekordtípus teszi lehetővé számunkra, hogy különböző adatok együttesét egyetlen logikai egységként kezeljünk. A rekord adattípus segítségével olyan programeszközöket tudunk deklarálni, amelyek egy adatbázistábla sorait közvetlenül tudják kezelni. Egy rekordtípus deklarációja a következőképpen történik:

```
TYPE név IS RECORD(
mezőnév típus [[NOT NULL] {:=|DEFAULT} kifejezés]
```

```
[,mezőnév típus [[NOT NULL] {:=|DEFAULT} kifejezés]]…);
```

A név a létrehozott rekordtípus neve, a továbbiakban deklarációkban a rekord típusának megadására szolgál. A mezőnév a rekord mezőinek, elemeinek neve.
A típus a REF CURSOR kivételével bármely PL/SQL típus lehet.
A NOT NULL megadása esetén az adott mező nem veheti fel a NULL értéket. Ha futási időben mégis ilyen értékadás következne be, akkor kiváltódik a VALUE_ERROR kivétel. NOT NULL megadása esetén kötelező az inicializálás.
A :=|DEFAULT utasításrész a mező inicializálására szolgál. A kifejezés a mező kezdőértékét határozza meg. Egy rekord deklarációjának alakja:

rekordnév rekordtípus_név;

```
DECLARE
TYPE cikk IS RECORD (
cikkkod NUMBER NOT NULL := 0,
cikknev VARCHAR2(20),
afa_kulcs NUMBER := 0.27
);
```

Az adattípusok között konverziós függvényeket használhatunk a konvertálásra. Ezeket a következő táblázat foglalja össze:

Függvény	**Leírás**	**Konvertálható családok**
TO_CHAR	A megadott paramétert VARCHAR2 típusúra konvertálja, opcionálisan megadható a formátum.	Numerikus, dátum
TO_DATE	A megadott paramétert DATE típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres
TO_TIMESTAMP	A megadott paramétert TIMESTAMP típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres

TO_TIMESTAMP_TZ	A megadott paramétert TIMESTAMP WITH TIMEZONE típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres
TO_DSINTERVAL	A megadott paramétert INTERVAL DAY TO SECOND típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres
TO_YMINTERVAL	A megadott paramétert INTERVAL YEAR TO MONTH típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres
TO_NUMBER	A megadott paramétert NUMBER típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres
TO_BINARY_DOUBLE	A megadott paramétert BINARY_DOUBLE típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres, Numerikus
TO_BINARY_FLOAT	A megadott paramétert BINARY_FLOAT típusúra konvertálja, opcionálisan megadható a formátum.	Karakteres, Numerikus
RAWTOHEX	A paraméterként megadott bináris érték hexadecimális reprezentációját adja meg.	Raw
HEXTORAW	A paraméterként megadott hexadecimális reprezentációjú értéket a bináris formájában adja meg.	Karakteres (hexadecimális reprezentációt kell tartalmaznia).
CHARTOROWID	A karakteres reprezentációjával adott ROWID belső bináris alakját adja meg.	Karakteres (18-karakteres rowid formátumot kell tartalmaznia).
ROWIDTOCHAR	A paraméterként megadott belső bináris reprezentációjú ROWID külső karakteres reprezentációját adja meg.	Rowid

Vezérlési szerkezetek

A feltételes utasítás

A feltételes utasítás egymást kölcsönösen kizáró logikai feltételek közül választ ki egyet, és ez alapján hajt végre egy vagy néhány utasítást.

Alakja:

```
IF feltétel THEN utasítás [utasítás]…
[ELSIF feltétel THEN utasítás [utasítás]…]…
[ELSE utasítás [utasítás]…]
END IF;
```

A feltételes utasításnak három formája van:

IF-THEN,

IF-THEN-ELSE és

IF-THEN-ELSIF.

A legegyszerűbb alak esetén a tevékenységet a THEN és az END IF alapszavak közé zárt utasítássorozat írja le. Ezek akkor hajtódnak végre, ha a feltétel értéke igaz. Hamis és NULL feltételértékek mellett az IF utasítás nem csinál semmit.

Az IF-THEN-ELSE alak esetén az egyik tevékenységet a THEN és ELSE közötti, a másikat az ELSE és END IF közötti utasítássorozat adja. Ha a feltétel igaz, akkor a THEN utáni, ha hamis vagy NULL, akkor az ELSE utáni utasítássorozat hajtódik végre.

A harmadik alak egy feltételsorozatot tartalmaz. Ez a feltételsorozat a felírás sorrendjében értékelődik ki. Ha valamelyik igaz értékű, akkor az utána következő THEN-t követő utasítássorozat hajtódik végre. Ha minden feltétel hamis vagy NULL értékű, akkor az ELSE alapszót követő utasítássorozatra kerül a vezérlés, ha nincs ELSE rész, akkor ez egy üres utasítás.

Az IF utasítás esetén bármely tevékenység végrehajtása után (ha nem volt az utasítások között GOTO) a program az IF-et követő utasításon folytatódik.

A THEN és ELSE után álló utasítások között lehet újabb IF utasítás, az egymásba skatulyázás mélysége tetszőleges.

```
declare
x number;
y number;
nagyobb number;
```

```
if y > x then nagyobb = y else nagyobb = x;
```

A CASE utasítás

A CASE egy olyan elágaztató utasítás, ahol az egymást kölcsönösen kizáró tevékenységek közül egy kifejezés értékei, vagy feltételek teljesülése szerint lehet választani. Az utasítás alakja:

```
CASE [szelektor_kifejezés]
WHEN {kifejezés | feltétel} THEN utasítás [utasítás]…
[WHEN {kifejezés | feltétel} THEN utasítás [utasítás]…]…
[ELSE utasítás [utasítás]…]
END CASE;
```

Ha a CASE utasítás címkézett, az adott címke az END CASE után feltüntethető.
Tehát egy CASE utasítás tetszőleges számú WHEN ágból és egy opcionális ELSE ágból áll. Ha a szelektor_kifejezés szerepel, akkor a WHEN ágakban kifejezés áll, ha nem szerepel, akkor feltétel.

Működése a következő:
Ha szerepel szelektor_kifejezés, akkor ez kiértékelődik, majd az értéke a felírás sorrendjében hasonlításra kerül a WHEN ágak kifejezéseinek értékeivel. Ha megegyezik valamelyikkel, akkor az adott ágban a THEN után megadott utasítássorozat hajtódik végre, és ha nincs GOTO, akkor a működés folytatódik a CASE utasítást követő utasításon.
Ha a szelektor_kifejezés értéke nem egyezik meg egyetlen kifejezés értékével sem és van ELSE ág, akkor végrehajtódnak az abban megadott utasítások, és ha nincs GOTO, akkor a működés folytatódik a CASE utasítást követő utasításon. Ha viszont nincs ELSE ág, akkor a CASE_NOT_FOUND kivétel váltódik ki.
Ha a CASE alapszó után nincs megadva szelektor_kifejezés, akkor a felírás sorrendjében sorra kiértékelődnek a feltételek és amelyik igaz értéket vesz fel, annak a WHEN ága kerül kiválasztásra. A szemantika a továbbiakban azonos a fent leírtakkal.

```
DECLARE
v_Allat VARCHAR2(10);
BEGIN
v_Allat := 'hal';
```

```
CASE v_Allat || 'maz'
WHEN 'halló' THEN
DBMS_OUTPUT.PUT_LINE('A halló nem is állat.');
WHEN SUBSTR('halmazelmélet', 1, 6) THEN
DBMS_OUTPUT.PUT_LINE('A halmaz sem állat.');
WHEN 'halmaz' THEN
DBMS_OUTPUT.PUT_LINE('Ez már nem fut le.');
ELSE
DBMS_OUTPUT.PUT_LINE('Most ez sem fut le.');
END CASE;
END;
```

Ciklusok

A ciklusok olyan programeszközök, amelyek egy adott tevékenység tetszés szerinti (adott esetben akár nullaszoros) ismétlését teszik lehetővé. Az ismétlődő tevékenységet egy végrehajtható utasítássorozat írja le, ezt az utasítássorozatot a ciklus magjának nevezzük.
A PL/SQL négyfajta ciklust ismer, ezek a következők:

- alapciklus (vagy végtelen ciklus);
- WHILE ciklus (vagy előfeltételes ciklus);
- FOR ciklus (vagy előírt lépésszámú ciklus);
- kurzor FOR ciklus.

A ciklusmag ismétlődésére vonatkozó információkat (amennyiben vannak) a mag előtt, a ciklus fejében kell megadni. Ezek az információk az adott ciklusfajtára nézve egyediek. Egy ciklus a működését mindig csak a mag első utasításának végrehajtásával kezdheti meg. Egy ciklus befejeződhet, ha

- az ismétlődésre vonatkozó információk ezt kényszerítik ki;
- a GOTO utasítással kilépünk a magból;
- az EXIT utasítással befejeztetjük a ciklust;
- kivétel váltódik ki.

Alapciklus

Az alapciklus alakja a következő:

```
[címke] LOOP utasítás [utasítás]…
```

```
END LOOP [címke];
```

Az alapciklusnál nem adunk információt az ismétlődésre vonatkozóan, tehát ha a magban nem fejeztetjük be a másik három utasítás valamelyikével, akkor végtelenszer ismétel.

Példa egy látszólag végtelen ciklus. De kivétel miatt ez is befejeződik, hiszen a faktoriális értéke meghaladja az 5 számjegyet.

```
DECLARE
v_Faktorialis NUMBER(5);
i PLS_INTEGER;
BEGIN
i := 1;
v_Faktorialis := 1;
LOOP
v_Faktorialis := v_Faktorialis * i;
i := i + 1;
END LOOP;
EXCEPTION
WHEN VALUE_ERROR THEN
DBMS_OUTPUT.PUT_LINE(v_Faktorialis
|| ' a legnagyobb, legfeljebb 5-jegyű faktoriális.');
END;
/
```

WHILE ciklus

A WHILE ciklus alakja a következő:

```
[címke] WHILE feltétel
LOOP utasítás [utasítás]…
END LOOP [címke];
```

Ennél a ciklusfajtánál az ismétlődést egy feltétel szabályozza. A ciklus működése azzal kezdődik, hogy kiértékelődik a feltétel. Ha értéke igaz, akkor lefut a mag, majd újra kiértékelődik a feltétel. Ha a feltétel értéke hamis vagy NULL lesz, akkor az ismétlődés befejeződik, és a program folytatódik a ciklust követő utasításon.
A WHILE ciklus működésének két szélsőséges esete van. Ha a feltétel a legelső esetben hamis vagy NULL, akkor a ciklusmag egyszer sem fut le (üres ciklus). Ha a feltétel a legelső

esetben igaz és a magban nem történik valami olyan, amely ezt az értéket megváltoztatná, akkor az ismétlődés nem fejeződik be (végtelen ciklus).

Az előző példát kivételkezelés nélkül is meg tudjuk oldani:

```
DECLARE
v_Faktorialis NUMBER(5);
i PLS_INTEGER;
BEGIN
i := 1;
v_Faktorialis := 1;
WHILE v_Faktorialis * i < 10**5 LOOP
v_Faktorialis := v_Faktorialis * i;
i := i + 1;
END LOOP;
DBMS_OUTPUT.PUT_LINE(v_Faktorialis
|| ' a legnagyobb, legfeljebb 5-jegyű faktoriális.');
END;
/
```

FOR ciklus

Ezen ciklusfajta egy egész tartomány minden egyes értékére lefut egyszer. Alakja:

```
[címke] FOR ciklusváltozó IN [REVERSE] alsó_határ..felső_határ
LOOP utasítás [utasítás]...
END LOOP [címke];
```

A ciklusváltozó (ciklusindex, ciklusszámláló) implicit módon PLS_INTEGER típusúnak deklarált változó, amelynek hatásköre a ciklusmag. Ez a változó rendre felveszi az alsó_határ és felső_határ által meghatározott egész tartomány minden értékét és minden egyes értékére egyszer lefut a mag. Az alsó_határ és felső_határ egész értékű kifejezés lehet. A kifejezések egyszer, a ciklus működésének megkezdése előtt értékelődnek ki.
A REVERSE kulcsszó megadása esetén a ciklusváltozó a tartomány értékeit csökkenően, annak hiányában növekvően veszi fel. Megjegyzendő, hogy REVERSE megadása esetén is a tartománynak az alsó határát kell először megadni:

```
FOR i IN REVERSE 1..10 LOOP …
```

A ciklusváltozónak a ciklus magjában nem lehet értéket adni, csak az aktuális értékét lehet felhasználni kifejezésben.
Ha az alsó_határ nagyobb, mint a felső_határ, a ciklus egyszer sem fut le (üres ciklus). A FOR ciklus nem lehet végtelen ciklus.

```
DECLARE
v_Osszeg PLS_INTEGER;
BEGIN
v_Osszeg := 0;
FOR i IN 1..100 LOOP
v_Osszeg := v_Osszeg + i;
END LOOP;
DBMS_OUTPUT.PUT_LINE(' 1 + 2 + ... + 100 = ' || v_Osszeg || '.');
END;
/
```

Az EXIT utasítás

Az EXIT utasítás bármely ciklus magjában kiadható, de ciklusmagon kívül nem használható. Hatására a ciklus befejezi a működését. Alakja:

```
EXIT [címke] [WHEN feltétel];
```

Az EXIT hatására a ciklus működése befejeződik és a program a követő utasításon folytatódik.

Vegyes tesztkérdések, feladatok

Vegyes feladatok

Adott a következő adattábla:

DOLGOZO(**id** integer not null primary key, **nev** varchar (50), **szdatum** date, **fizu** numeric(12,2), **sz_hely** varchar (30), **belep_ev** int, **neme** char(1));

Id: azonosító, nev: a dolgozó neve, szdatum: születési dátum, fizu: a dolgozó fizetése, sz_hely: a város, ahol született, belep_ev: a céghez belépés éve, neme: F-férfi, N-nő.

Válassza ki a helyes SQL mondatokat, melyek megadják a választ a feltett kérdésekre!

Egy kérdésre legalább egy jó választ kell kiválasztani.

1. Minden Kovács adata:

A. SELECT * FROM dolgozo WHERE nev LIKE „Kovács%”;
B. SELECT * FROM dolgozo WHERE nev = „Kovács%”;
C. SELECT * FROM dolgozo WHERE nev = Kovács ;
D. SELECT * FROM dolgozo WHERE nev LIKE Kovács%;

2. A 100 000 és 120 000 Ft közötti fizetések:

A. SELECT nev, fizu FROM dolgozo WHERE fizu BETWEEN 100000 AND 120000;
B. SELECT nev, fizu FROM dolgozo WHERE fizu BETWEEN „100000 Ft” AND „120000 Ft”;
C. SELECT nev, fizu FROM dolgozo WHERE (fizu >= 100000) AND (fizu <= 120000);
D. SELECT nev, fizu FROM dolgozo WHERE (fizu >= 100000) OR (fizu <= 120000);

3. Az 1970.01.01 után született dolgozók neve és fizetése, akik 100000 Ft-nál többet keresnek.

A. SELECT nev, fizu FROM dolgozo WHERE (sz_datum > ’1970.01.01’) OR (fizu > 100000);
B. SELECT nev, fizu FROM dolgozo WHERE sz_datum > ’1970.01.01’ AND fizu > 100000;
C. SELECT nev, fizu FROM dolgozo WHERE sz_datum IN ’1970.01.01’ AND fizu < 100000;
D. SELECT DISTINCT nev, fizu FROM dolgozo WHERE sz_datum > ’1970.01.01’ AND fizu < 100000;

4. Városonként az ott született dolgozó száma

A. SELECT sz_hely AS Hely, COUNT(id) as Törzsszám FROM dolgozo ORDER BY sz_hely;
B. SELECT sz_hely AS Hely, COUNT(id) as Törzsszám FROM dolgozo Where sz_hely in dolgozo;
C. SELECT sz_hely AS Hely, COUNT(id) as Törzsszám FROM dolgozo GROUP BY sz_hely;
D. SELECT sz_hely AS Hely, SUM(id) as Törzsszám FROM dolgozo GROUP BY sz_hely;

5. Melyek azok a városok, ahol a dolgozók átlagfizetése kisebb mint 120 000 Ft.

A. SELECT sz_hely AS Hely, AVG(fizu) as átlag FROM dolgozo GROUP BY sz_hely HAVING fizu < 120 000;
B. SELECT sz_hely AS Hely, AVG(fizu) as átlag FROM dolgozo Where fizu < 120 000 GROUP BY sz_hely ;
C. SELECT sz_hely AS Hely, AVG(fizu) as átlag FROM dolgozo GROUP BY sz_hely HAVING AVG(fizu) < 120 000;
D. SELECT nev AS Hely, AVG(fizu) as átlag FROM dolgozo GROUP BY sz_hely HAVING AVG(fizu) < 120 000;

6. Listázza ki a tavaly belépett nőket névsor szerint csökkenő sorrendben!

.....
.....
.....
.....

7. A legalább 5 éve itt dolgozó férfiak közül kik keresnek az átlagnál többet?

.....
.....
.....
.....

8. Hány férfi és hány nő született Egerben?

.....
.....
.....

.....

9. A dolgozók 10%-al emelt fizetését ki tudjuk listázni a jelenlegi fizetés mellé, a dolgozó nevével, mert a lekérdezés az adatokat több táblából is össze tudja válogatni.

A. Igaz-Igaz-van összefüggés B. Igaz-Igaz-Nincs összefüggés

C. Igaz-Hamis-Nincs összefüggés D. Hamis-Igaz-Nincs összefüggés

D. Hamis-Hamis-Nincs összefüggés

10. A having záradék a csoportosított rekordokat szűri tovább, mert a having nagyobb prioritással rendelkezik, mint a where.

A. Igaz-Igaz-van összefüggés B. Igaz-Igaz-Nincs összefüggés

C. Igaz-Hamis-Nincs összefüggés D. Hamis-Igaz-Nincs összefüggés

D. Hamis-Hamis-Nincs összefüggés

11. A tulajdonság előfordulás az adattábla egy …

A. Rekordja C. Mezője

B. Mezőérték D. Mezőtípusa

12. A módosítási anomália azt jelenti hogy …

A. Rekord módosításkor más rekord módosítása is szükségessé válik.

B. Rekord módosításkor a módosítás sikertelen lesz.

C. Rekord beszúrásakor más rekord beszúrása is szükségessé válik.

D. Rekord beszúrásakor a beszúrás sikertelen lesz.

13. Melyek egész számokat tartalmazó típusok?

A. SmallInt B. Integer

C. BigInt D. Char

14. Melyek valós számokat tartalmazó típusok?

A. VarChar B. Real

C. Boolean D. Float

15. A fizetes táblában a 10%-os fizetésemelést így végezzük el minden dolgozóra:

A. insert fizetes set fiz = 1.1*fiz;

B. Update fizetes set fiz = 1.1*fiz;

C. Update fizetes from fiz = 1.1*fiz;

D. Update fizetes set fiz = 10% * fiz;

16. A Kis nevű dolgozók városa legyen Eger:

A. Update dolgozo set varos = ’Eger’ having nev like ’Kis’;

B. Update dolgozo from varos = ’Eger’ where nev like ’Kis’;

C. Update dolgozo set varos = ’Eger’ where nev like ’Kis’;

17. Milyen záradékkal tudunk rendezni?

A. Where B. Broup by

C. Order by D. Having

18. Mivel tölti ki az üres helyeket a fix hosszúságú szöveg esetén a rendszer?

A. Semmivel B. Szóközzel

C. Aláhúzással D. #9 kódú karakterrel

19. Hogyan hozzuk létre a *tanuló* táblát?

A. create new table tanulo (id int primary key, nev char(30));

B. alter table tanulo add nev char (30);

C. create table tanulo (id int primary key, nev char(30));

D. alter table tanulo tanulo (id int primary key, nev char(30));

20. Írja le a 3. normálforma definícióját, definiálja az ehhez szükséges fogalmat is.

.....

.....

.....

.....

.....

.....

.....

.....

21. . Írja le SQL parancsokkal a következő lépéseket:

a. Hozza létre a STUDENT táblát, amiben szerepel az ID egész típusú mező, mint kulcs, a hallgató neve, születési dátuma.

b. Szúrja be új rekordként 17 azonosítóval Kiss Ferenc 1983. március 19-én született hallgató adatait.

c. Javítsa a 17-es azonosítójú hallgató nevét Kiss Mátyásra.

d. Bővítse a táblát a CITY mezővel, ahol a születési helyet fogja tárolni.

22. Válaszoljon a következő kérdésekre egy-egy lekérdezéssel:

a) Készítsünk név szerint rendezett listát, azon tanulókról, akik Budapesten születtek és valamelyik keresztnevük Zsolt.

b) Számoljuk ki születési helyenként az átlagos befizetendő menzapénzt, listázzuk születési hely szerint csökkenő sorrendben.

c) Listázzuk ki az átlagos menzapénznél többet fizetők nevét és születési dátumát és menzapénzét, a pénz szerint növekvő sorrendben!

d) Tegyük fel hogy létezik a STUDENT tábla mellett a következő tábla:

BEFIZ(ssz int PK, osszeg int, tanulo int, datum date)

Listázzuk ki tanulónként az összes befizetett összeget, név szerint növekvő sorrendben.

e) Készítsünk listát a budapesti tanulók befizetéseiről, melyek 2009 utolsó hónapjában történtek.

23. Alaptáblák

ember [id integer primary key, nev varchar(40) not null, varos varchar(40)]

auto [rsz char(7) primary key, tulaj integer, tipus varchar(20), szin varchar(20), ar numeric(7,0)]

Hozzul létre a két adattáblát.

Feladatok:

1. Kérdezzük le a piros színű autók árait.
2. Az 500000 és 1000000 Ft közötti értékű autók árát növeljük 20%-al.
3. Kérdezzük le a ’K’ betűvel kezdődő tulajdonosokat és autóik típusát.
4. Kérdezzük le a miskolci és egri autótulajdonosok nevét és autóik árát, a tulajdonosok szerint névsorrendbe rendezve.

5. Kérdezzük le azoknak az autótulajdonosoknak a nevét, akiknek 1 millió Ft-nál olcsóbb autójuk van.
6. Kérdezzük le azon autótulajdonosoknak a nevét és címét, akiknek van autójuk.
7. Azoknak az autóknak a rendszámát kérdezzük le, amelyeknek miskolci a tulajdonosuk.
8. Kérdezzük le azokat az autókat, amelyeknek az ára nagyobb minden piros autóénál.
9. Kérdezzük le, hogy milyen típusú autó fordulnak elő az autó táblában (ismétlődés nélkül).
10. Kérdezzük le a miskolci autók átlagárát.
11. Kérdezzük le, hogy az egyes városokban hány autó van.
12. Kérdezzük le az átlagárnál drágább autók rendszámát és tulajdonosaik nevét.
13. Kérdezzük le a legolcsóbb miskolci autónál drágább autók rendszámát.

24. Hozza létre az alábbi szerkezetű, DOLGOZOK nevű táblát:

KOD VARCHAR2(4) NOT NULL
NEV VARCHAR2(30) NOT NULL
FIZETES NUMBER
SZUL DAT DATE

25. Bővítse a DOLGOZOK táblát a CIM oszloppal, melynek típusa VARCHAR2(30)! A NEV hosszát módosítsa 40-re!

26. Hozza létre az UJ_RESZL1 nevű táblát, melynek szerkezete azonos a RESZLEG nevű tábla szerkezetével!

27. Hozza létre az UJ_RESZL2 nevű táblát, melynek szerkezete és tartalma azonos a RESZLEG nevű tábla szerkezetével és tartalmával!

28. Nevezze át az UJ_RESZL2 táblát RESZLEG2 névre!

29. Hozza létre azt a NEZET nevű nézettáblát, amely az ALKALMAZOTT és a RESZLEG táblából csak az 'ELADO' beosztású dolgozók kódját, nevét, beosztását, részlegének kódját, nevét és címét tartalmazza!

30. Hozza létre azt a VIDEK nevű nézettáblát, amely csak a nem Budapesti részlegek adatait tartalmazza!

31. Hozza létre az ATLAG nevű nézettáblát, amely a részlegek kódját és az ott dolgozók átlagfizetését tartalmazza! Készítsen listát a létrehozott nézettábla segítségével, amelyben a dolgozók neve, fizetése, részlegének kódja és a részleg átlagfizetése szerepel!

32. Az előző feladatban létrehozott ATLAG nevű nézettábla felhasználásával írassa ki a dolgozók nevét, fizetését, részlegének nevét és címét, valamint a részlegben dolgozók átlagfizetését!
33. Készítse el a RENDELES és AUTOK táblák alapján az UJ_RENDELES nézettáblát, melynek oszlopai az ügyfélszám, az autócsoport, a kölcsönzött autó típusa, a rendelés dátuma, a megrendelő személy neve, a kölcsönzési idő kezdete és időtartama, valamint a fizetés módja legyenek! Listázza ki a tábla tartalmát!
34. Módosítsa az előző feladatban létrehozott nézettábla szerkezetét úgy, hogy egy oszlop a kölcsönzési idő alatt futott km-ek számát is mutassa!
35. Hozza létre az UGYFELEK, TIPUSOK, AUTO_CSOP és RENDELES táblák alapján azt az UJ_UGYFEL nevű nézettáblát, amely a következő oszlopokat tartalmazza: az ügyfél száma, neve, a kapcsolatot tartó személy neve, a rendelt autó típusa, rendszáma, rendelési ideje, a kölcsönzés ideje alatt futott km, a kölcsönzési díj km-enként és naponként!
36. Hozza létre a KOLCSON_SZAM nevű nézettáblát a RENDELES tábla alapján, amely rendszámonként tartalmazza a kölcsönzések számát! Irassa ki a nézettábla tartalmának felhasználásával az AUTOK táblában szereplő autók rendszámát, típusát és a kölcsönzések számát! Az egyszer sem kölcsönzött autóknál a kölcsönzések száma 0 legyen!
37. Hozza létre azt a táblát, amely a rendszámot, az utolsó szerviz idején mutatott és a jelenlegi km-óra állást, valamint a kötelező szerviz intervallumot tartalmazza! A tábla neve KARBANTART legyen!
38. Az AUTOK tábla alapján hozza létre az ELADO_AUTOK nevű táblát, melynek oszlopai a rendszám, a típusnév, a vásárlás dátuma és a futott kilométerek száma legyen (új oszlopnevekkel)!
39. A KARBANTART nevű táblában növeljük meg a kilométeróra állását mutató oszlop hosszát 8-ra!
40. Bővítse a KARBANTART táblát a következő szerviz nevű oszloppal! Hossza legyen 8, típusa numerikus!
41. Hozzon létre indexet az AUTOK táblára a rendszám alapján!
42. Hozzon létre indexet a RENDELES táblához az ügyfélszám és az autó típus név szerint!
43. Vigye fel a RESZLEG2 táblába a 80-as kódú részleg adatait! Részleg név: AUTOKOLCSONZO, cím: SZEGED.
44. Vigye fel a RESZLEG2 táblába a RESZLEG táblából a kölcsönző irodák adatait!

45. Vigye fel a VIDEK nézettáblába a 99-es kódú részleget FORD --- AUTO névvel, DEBRECEN címmel, majd nézze meg, hogy bekerült-e a sor a RESZLEG táblába és a nézettáblába!

46. Módosítsa a RESZLEG2 táblában a 'KOZPONT' részleg nevét 'IRODAK'-ra!

47. Növelje meg a 10-es kódú részlegben dolgozók fizetését 15%-kal!

48. Növelje meg az'ELADO' beosztású dolgozók prémiumát 10000 Ft tal!

49. Törölje ki a RESZLEG2 táblából a debreceni részlegeket!

50. Vigye fel az AUTOK táblába a legújabban beszerzett autó adatait:

Rendszám: CAR-342

Típusnév: RENAULT ESPACE

Autócsoport: LUXUS

Vásárlás dátuma: 1994. június 23.

Ár: 1.400.000 Ft

Futott km: 100

Utolsó szervíz: 0 km-nél

Állapot: kiadható (A)

Részleg: 20

51. Vigye fel az ELADO_AUTOK táblájába azokat az autókat, amelyek 150000-nél többet futottak!

52. Módosítsa az AUTO_CSOP táblában a kölcsönzés km-enkénti díját 10%-kal megnövelt értékre!

53. Módosítsa a NORMAL autócsoport autóira a szervízelések közötti intervallumot 12000 km-rel

54. Törölje az AUTOK táblából azokat az autókat, melyeket az ELADO_AUTOK táblába felvitt! (175. feladat)

55. Törölje az AUTOK táblából az ABC-022 rendszámú autót!

56. Írjon INSERT utasítást, amely az EXTRA autócsoport valamennyi autójának adatait beírja a most létrehozott, de még üres EX AUTOK táblába!

57. Írjon UPDATE utasítást, amely az EX_AUTOK tábla valamennyi 'OPEL ASTRA' típusú autó részleg kódját'99'-re módosítja!

58. Törölje az EX AUTOK tábla tartalmát a'99'-es kódú autók kivételével!

59. Vigye át az AUTOK táblából az ELADO_AUTOK táblába azon járműVek adatait, amelyek az autócsoportjuk átlagánál 50%-kal több kilométert futottak!

60. Új ügyfelet kell felvennie az UGYFELEK táblába!

(Hozzon létre egy szekvenciát 351-es kezdőértékkel!)

Ügyfélszáma: a táblában következő érték

Neve: Karát KFT.

Címe: 4025 Debrecen, Nyugati utca 7.

Megbízott: Nagy Péterné

A cég átutalással fizet.

Idézett forrásmunkák *(lehetőleg élő hivatkozással)*

Szerző. (2012). *Könyv hivatkozás.* Eger: EKF.

Irodalomjegyzék *(lehetőleg élő hivatkozással)*

Szerző. (2012). *Könyv hivatkozás.* Eger: EKF.

1. Jeffrey D. Ullman – Jennifer Widom: *Adatbázisrendszerek –alapvetés* 19-25p., 1998. Panem