

BÓDY BENCE

**Az SQL
példákon keresztül**

JEDLIK OKTATÁSI STÚDÓ
Budapest, 2003

J.O.S.

Minden jog fenntartva. Ezt a könyvet vagy annak részleteit a Kiadó engedélye nélkül bármilyen formában vagy eszközzel reprodukálni, tárolni és közölni tilos.

A könyv készítése során a Kiadó és a Szerző a legnagyobb gondossággal jártak el. Az esetleges hibákat és észrevételeket a jos@jos.hu e-mail címen szívesen fogadjuk.

Lektorálta:

Kupcsikné Fitus Ilona, főiskolai docens

Borító: Sarkadi Csaba, 2003

© Kiadó: Jedlik Oktatási Stúdió Bt.

1212 Budapest, Táncsics M. u. 92.

Internet: <http://www.jos.hu>

e-mail: jos@jos.hu

Felelős kiadó: a Jedlik Oktatási Stúdió Bt. cégvezetője

Nyomta: LAGrade Kft.

Felelős vezető: Szutter Lénárd

ISBN: 963 210 860 4 Raktári szám: JO 0311

Tartalomj egyzék

Tartalomjegyzék.....	5
Előszó.....	7
Bevezetés	9
A modellezendő valóság	13
Adatmodellezés	14
Az adatbázis bemutatása	24
Kidolgozott feladatok.....	28

Feladatok:

1. ÉS feltétel	35
2. VAGY feltétel	38
3. A logikai műveletek sorrendje	41
4. A feltétel tagadása	44
5. Egyszerű lekérdezések egy táblából	46
6. Paraméteres lekérdezés	54
7. Táblák összekapcsolása, direktzorzat	56
8. Beágyazott SELECT utasítás	62
9. Csoportok kiválogatása	67
10. Maximumkeresés.....	74
11. Csoportonkénti maximum keresése	83
12. Adatbázisbeli megszorítások ellenőrzése	87
13. Két szempont szerinti csoportosítás.....	90
14. Az eredménytáblát szűkítő feltételek	92
15. N : M viszonyú táblák összekapcsolása.....	96
16. Legyártható-e egy termék?	98
17. Adott időintervallumban megrendelt, illetve meg nem rendelt termékek.....	105
18. Egy adott termékcsoportot megrendelt megrendelések listája.....	110
19. Az ALL, ANY és EXISTS predikátum összehasonlítása	113
20. Csoportokba sorolás külső feltételek szerint	119
21. Sávosan adott kedvezmény	125
22. Az eredménytábla sorainak sorszámozása	134
23. Előre meghatározott sorú táblázat készítése	137
24. Számítások a SELECT utasítás mezőlistájában	141
25. Azonos anyagokból felépülő termékpárok kikeresése	147
26. Különböző szerkezetű sorokból álló lista	159
27. Összegfokozatos lista	162

Tartalomjegyzék

28. Külső ciklusváltozó szerintiválogatás	168
29. Hiányzó értékek megkeresése folytonos és nem folytonos mezőértékekből	170
30. Az árak változásának követése.....	179
 Gyakorló feladatok.....	 187
Access függvények	189

Előszó

Évek óta tanítok különböző intézményeknél adatbázis-tervezést, adatbázis-kezelést és ezen belül tanítom az SQL nyelv használatát. Több jó jegyzettel, könyvvel találkoztam, de úgy tapasztaltam, a kereskedelemben nem kapható olyan könyv, amelyikből egy átlagos hallgató megismerhetné az SQL nyelv szépségeit, hatékonyságát.

Amikor az olvasó kezébe veszi ezt a könyvet, először azt kell tisztázni, hogy mire számíthat, mire terjed ki és mire nem terjed ki ez a példatár. A könyv címe is arra utal, hogy ez a könyv elsősorban példatár, a célja az „SQL gondolkodásmód” bemutatása sok példával, és mindegyik példának többféle megoldásával. Nem célja az egyes SQL implementációk bemutatása, így a megoldások hatékonyságával sem foglalkozom. Bár lehetne általános érvényű szabályokat mondani, azonban ezek csak részizgazságok lennének. A konkrét feladat és a konkrét erőforrások ismeretében tudjuk a lehetséges megoldások közül a leghatékonyabb megoldást kiválasztani. Választani akkor tudunk, ha több megoldási lehetőséget ismerünk, és ebben akar segítséget nyújtani a példatár.

Modellezési kérdésekkel csak annyira foglalkozom, amennyit a példatárban tárgyalt egyetlen adatbázis kialakítása, megértése megkíván. A relációs adatbázis-szemlélet matematikai megalapozása is hiányzik a könyvből. Ez a példatár kiegészítése azoknak a könyveknek, amelyek az elméleti megalapozást részletesen tárgyalják, de a feladatok megoldásában az olvasókat már magukra hagyják.

A példatár nem tételez fel semmilyen előzetes ismeretet az SQL nyelvről, azonban nem írom le pontos definíciókkal az egyes SQL záradékok szintakszisát és szemantikáját, meggyőződésem, hogy a példákhoz fűzött magyarázatokból ezek egyértelműen kiderülnek.

Az SQL a relációs adatbázis-kezelés szabványos nyelve, amely kiterjed az adatbázisséma definiálására, az adatok aktualizálására és lekérdezésére. A példatár ebből a körből csak a lekérdezésekkel foglalkozik, pontosabban a SELECT utasítással és annak a záradékaival. Az SQL szemléletmód megértésének a kulcsa a SELECT utasítás rejtelmeinek az elsajátítása. A sémadefiniáló rész, az adatok aktualizálása, bár szabványos, általában környezetfüggő, így inkább az adott rendszer kézikönyvébe tartozó téma, mint egy ilyen példatárba.

A példákat Microsoft ACCESS 2002-ben ellenőriztem ANSI 89 SQL beállítással. A példatár nem ACCESS jegyzet, nem foglalkozik az ACCESS lehetőségeinek az ismertetésével, azonban, mivel a legtöbb olvasónak a legkönnyebben elérhető SQL felületű adatbázis-kezelő az ACCESS, azért, hogy a példákat az olvasó saját maga is ki tudja próbálni, a gyakorló feladatokat önállóan meg tudja oldani, az ACCESS lekérdezés rácsáról és SQL felületéről a cél érdekében szükséges, minimális információt tartalmazza a könyv.

Előszó

Összefoglalva, kinek is szól ez a példatár? Mindenkinnek, aki SQL-t tanul, vagy aki már tanult, de szeretné megismerni jobban az SQL lekérdezés lehetőségeit. Mindazoknak, akik élvezni tudják, hogy egy gondolatot milyen sokféleképpen lehet megfogalmazni. Meggyőződésem, hogy olyanok is találhatnak hasznos ötleteket a példák megoldásai között, akik a munkájuk során már évek óta használnak SQL lekérdezéseket.

Végezetül szeretnék köszönetét mondani Kupcsikné Fitus Ilona főiskolai docensnek, a SZÁMALK Oktatási Rt. oktatójának, akik nagyon sokat segítettek abban, hogy ez a könyv elkészülhetett, és mint a könyv lektora, értékes ötleteivel gyarapította a példatárat. Köszönöm továbbá a hallgatóknak azt a sokféle megoldásoldást, amit néha egy egyszerű lekérdezésre is képesek voltak adni. Sok ötletet merítettem ezekből a dolgozatokból is.

Budapest, 2003. április 30

Bódy Bence
szerző

Bevezetés

SQL történelem

E. F. Codd, az IBM amerikai kutatója, az 1970-ben megjelent cikkével (A Relational Model of Data for Large Shared Data Banks) megalapozta az adatfeldolgozás és tárolás relációs szemléletének a kialakítását.

A relációs szemléletben az adatok kétdimenziós adattáblákban kerülnek tárolásra. Több adattábla együttesen alkotja a relációs adatbázist.

Codd által kidolgozott adatbázis-modell talán legfontosabb eredménye, hogy az adatfeldolgozás hagyományos modelljével szemben, amelyik az adatfeldolgozás eredményét az eredményekhez vezető algoritmussal írta le, a feldolgozás eredményének a meghatározására fektette a hangsúlyt. A kiinduló táblázatokból az eredményt a táblákkal végzett műveletekkel írta le. A célhoz vezető úttól, az adatok fizikai szerkezetétől, szervezési és tárolási módjától független lekérdezési módszert dolgozott ki. A másik eredmény a modell kidolgozásához felhasznált eszközökből következett. Codd a modelljét matematikai eszközökkel, n -ed fokú relációk fogalomrendszerével írta le. A modell segítségével az adatbázisokra olyan fogalmakat tudott pontosan definiálni, mint redundancia, konzisztencia, függőségi viszonyok, normálformák, adatintegritás biztosítása, adatstruktúra definiálása, adatok aktualizálása (felvitel, módosítás, törlés), lekérdezés stb.

Codd a cikkében két lekérdező nyelvet definiált, az egyik az adattáblákat, mint halmazokat tekinti, és halmazműveletekkel kapja meg a kívánt eredményt, a másikban, logikai állításoknak tekinti a táblák sorait, és logikai kifejezésekkel írja le az eredménytáblát. A két nyelvről kimutatta, hogy azonos erősségűek.

Az IBM 1974-ben kidolgozta a System/R rendszert, amely a relációs adatbázis-kezelés prototípusának tekinthető. 1976-ban definiálták a System/R rendszer több felhasználós, kétdimenziós táblázatokba szervezett adatokat lekérdező nyelvét, a SEQUEL (Structured English Query Language) nyelvet.

A projekt folytatásaként 1979-ben megszületett az első, kereskedelmi forgalom számára készített relációs adatbázis-kezelő rendszer, amelyet Oracle rendszernek neveztek el, és először használták az SQL (Structured Query Language) kifejezést, mint az adatbázis lekérdező nyelvet.

1980-ban megjelent az Ingres relációs adatbázis-kezelője QUEL nyelvvel.

Az IBM sem tétlenkedett, 1983-ban jelentette be az SQL/Data System-et, rövid nevén a DB2 rendszert. Lényegében ez az SQL nyelv lett a későbbi szabvány alapja.

1986-ban az Ingres is áttért az SQL használatára.

1986-ban ANSI (American National Standards Institute) elfogadta az első SQL szabványt. A következő, az 1989-es SQL szabvány az előfordítók és a dinamikus

SQL ajánlásaira is kitért. 1992-ben a szabvány további elemekkel bővült. Szabványosították a programozó által definiálható adattípusokat, megszorító szabályok megadási lehetőségét, kulcsdefiníciókat és kulcsvizsgálatokat, tárolt eljárások megadását, stb. Ezzel lezárult az adatbázis-kezelés szabványosításának egy fejezete, de természetesen nem véglegesen. Körvonalazódik már a fejlődés további útja, létezik már az objektum orientált szemlélet betörése az adatbázis-kezelés területére is, de ez még nem annyira kiforrott, hogy szabványosítani lehessen.

A szabványosítás következménye, hogy napjainkban az SQL a relációs adatbázis-kezelés általánosan elfogadott nyelvéné vált. Ma minden jelentősebb adatbázis-kezelő rendszer SQL alapú, köztük van az IBM DB2, a Microsoft SQL Server 7, az Oracle 8, az Informix, a Sybase, hogy csak a legnagyobbakat említsük. Az egyes megvalósítások között vannak kisebb eltérések, de a legfontosabb kérdésekben, a nyelv „filozófiájában” nincs eltérés.

Az SQL tartalmaz adatstruktúra definiáló, adatokat aktualizáló (felvitel, módosítás, törlés), lekérdező, adatintegritás biztosító, adatbázis tranzakció kezelő utasításokat, azonban az egyes implementációk a szabványtól több-kevesebb eltérést tartalmaznak, ezért a pontos definíciók megismeréséhez feltétlenül szükség van az adott rendszer kézikönyvére.

A példatár felépítése

A példatárban egyetlen, a valósághoz képest leegyszerűsített adatbázison keresztül mutatjuk be az SQL utasításokat. Az első fejezet a feladat leírását, a modellezendő valóságot tartalmazza. A második fejezetben egy keveset foglalkozunk adatmodellezéssel. Feltételezzük, hogy az olvasó ismeri ezt a területet, ha mégsem, feltétlenül javasoljuk, hogy nézzen utána. A harmadik fejezetben bemutatjuk a konkrét adatbázist, amelyet a továbbiakban használni fogunk. Nem, illetve csak minimálisan foglalkozunk a tárolt adatok tulajdonságaival, az adatok megjelenítését szabályzó formázási műveletekkel.

A példatár az SQL utasítások közül csak a SELECT utasítást tárgyalja, ACCESS kifejezéssel csak a választó lekérdezést. Továbbá példát mutatunk olyan lekérdezésre is, amikor két tábla UNION egyesítésével kapjuk az eredménytáblát.

Javasoljuk az olvasónak, hogy építse fel valamilyen rendszerben az adatbázist és töltsen fel a táblákat néhány sorral. Egyszerre csak egy fejezetet olvasson végig, majd gépelje be az SQL utasításokat, és ellenőrizze az eredményt. Az egyes fejezetek önmagukban is érthetőek, nem épülnek egymásra. Azoknak, akik már ismerik valamennyire az SQL utasításokat, azoknak is javasoljuk, hogy ne lapozzák át az elejét, azt is olvassák végig. Minden fejezet végén található a fejezetben bemutatott és kidolgozott példákhoz hasonló, néhány további feladat. Ezek

között a gyakorló feladatok között vannak bonyolultabb feladatok is, de mindig a fejezethez kapcsolódó ötletek továbbfejlesztésén alapulnak. Ha az olvasó ellenőrizni akarja, hogy tényleg megértette a fejezetben kidolgozott példákat, próbálja megoldani ezeket a feladatokat is.

A példatár végén van néhány további feladat. Ezeket már csak a tartalmuk és nem a megoldásukhoz szükséges SQL utasítások szerint csoportosítottuk.

Eltérések a szabványtól

A példatárban a feladatokat Microsoft ACCESS 2002 szintaxis szerint ANSI 89 SQL beállítással írtuk. Az ACCESS néhány fontos eltérése az SQL szabványtól:

- a LIKE összehasonlító operátornál az ACCESS a WINDOWS operációs rendszernek megfelelő ? és * karaktert használja a szabványos _ és % karakter helyett,
- hiányzik a COUNT(DISTINCT mezőnév), de megmutatjuk, hogyan lehet ezt helyettesíteni más SQL utasításokkal,
- a dátum konstansokat # karakterek közé kell tenni a { } jelek helyett,
- a karakteres konstansnál megengedi mind a nem szabványos " jel, mind a szabványos 'jel használatát,
- a karaktersorozatok összeépítésénél mind az & jel, mind a + jel megengedett,
- megengedett a magyar WINDOWS rendszerben az ékezetes karakterek használata.

Ha valaki más rendszerben, vagy nem magyar WINDOWS operációs rendszerben próbálja a feladatokat megoldani, először ellenőrizze, hogy abban a környezetben milyen eltérések vannak a szabványos SQL-től, és elfogadja-e a példatárban használt ACCESS jelöléseket.

A példatárban használt jelölések

Reláció, adat- illetve eredménytábla jelölése:

TÁBLANÉV{mező1, mező2, ... mezőn}

- a tábla neve mindig nagybetű,
- a mezőneveket kisbetűvel írjuk, bár a mezőnév több szóból is állhat, általában egyszavas mezőneveket használunk,
- a kulcsmezőket aláhúzással jelöljük.

Az SQL utasításokban a soremelésnek, felesleges betűközöknek, tabulátomak nincs jelentősége. Az SQL utasítás végét pontosvessző jelzi. A jobb olvashatóság érdekében az egyes záradékokat új sorba és tagolva írjuk, pl.:

```
SELECT Left(kód,1), kód, MIN(rend_ár)
FROM rend_ár
GROUP BY kód
HAVING MIN(rend_ár) > 10000;
```

Az SQL utasításban a kis- és nagybetű használatának nincs jelentősége, de a példatárban a következő jelölést használjuk:

- minden SQL fenntartott szót nagybetűvel írunk,
- a mezőneveket, táblaneveket kisbetűvel írjuk,
- a felhasznált ACCESS függvények nevét az ACCESS súgóban megtalálható formában írjuk.

```
SELECT kód, azonosító, mennyiség
FROM szerkezet
WHERE kód IN (SELECT kód
FROM szerkezet
WHERE azonosító = 113);
```

Táblánál minősítő nevet (alias) csak akkor használunk, ha szükség van rá, de az AS elhagyható kulcsszót nem használjuk, például:

```
SELECT első.rend_szám
FROM rendelés első, rendelés másod
WHERE első.kód = 'BC1' AND másod.kód = 'BC9'
AND első.rend_szám = másod.rend_szám;
```

Amennyiben egy lekérdezés eredményéből további lekérdezések szeretnénk végrehajtani, a lekérdezést (nem az eredménytáblát) el tudjuk tenni minden SQL implementációban. A lekérdezés eredménytáblájáról beszélünk, de ez nem jelenti a tábla fizikai eltevéését. Az ACCESS a lekérdezés bezárásánál kérdez rá, hogy akarjuk-e és milyen névvel tárolni a lekérdezést. A lekérdezés tárolását a példatárban így jelöljük:

-> RENDELT{partner_kód, kód}

A lekérdezés eredményében nem mindig van kulcsmező, de ha van, akkor sem jelöljük.

A modellezendő valóság

A példatárban egyetlen feladat köré csoportosítva fogjuk ismertetni az SQL lekérdezés lehetőségeit. A feladat természetesen nem valóságos, csak annyit és úgy veszünk át egy valódi alkalmazásból, amennyire az SQL megismerése szempontjából szükség van.

A kiinduló feladat tehát a következő:

Egy cég megrendelésre termékeket szerel össze a raktárán lévő alkatrészekből és anyagokból.

Egy megrendelés több termékre is vonatkozhat, és mindegyik termékből több darabot is megrendelhetnek egyszerre. Tárolják a megrendelés dátumát is.

A megrendelés elfogadása előtt megvizsgálják, hogy van-e elegendő alkatrész, illetve anyag a raktáron, hogy a kívánt termékből a megrendelt darabszámot összeállítsák, és ha a termék legyártható, elfogadják a megrendelést, majd a megfelelő raktári tételeket csökkentik. A megrendelés elfogadásánál minden termékre külön megállapítják az elkészítés határidejét. Ha a termékből több darabot rendeltek meg, akkor is csak egy elkészítési dátumot határoznak meg. Rendelésenként van részszállítás, de termékenként nincs.

Ha a termékből a kívánt darabszám elkészült, az adatbázisban a kész mezőt igen-re állítják.

Egyszerűsítések

Az anyagokat és alkatrészeket nem különböztetjük meg, úgy tekintjük, mintha csak anyagokból állna egy termék. Az anyagok mértékegysége utal az anyag felhasználási egységére. Nyilván, ha a mértékegységhez darabszámot írtunk be, értelmetlen nem egész értékű felhasználást figyelembe venni, azaz nem anyagról, hanem beépítendő alkatrészből van szó, de ezt a különbséget a lekérdezésekben nem fogjuk kihasználni. Nincs beépítési hierarchia, azaz nincsenek szerelvények, nincsenek félkész termékek. Még egy, nem túl lényeges megszorítás, minden termék legalább egyféle anyagból áll.

A feladatban nem foglalkozunk a megrendelő partnerek adataival, nem foglalkozunk továbbá a beszállítással, a számlakészítéssel. A termékek áránál nem kezeljük az ÁFÁ-t.

Adatmodellezés

Nem célja ennek a példatárnak, hogy az adatmodellezés érdekes és hasznos tárgykörét részletesen ismertesse, de a fejezet két - inkább gyakorlati, mint elméleti - normalizációs technikát bemutat. A következő fejezet nem törekszik matematikai alaposságra, inkább tekinthető a témához kapcsolódó „mesé”-nek.

I. Problémakör

Tegyük fel, hogy a könyvben használt feladatban a termékek megrendelése mindig egy meghatározott formájú bizonylaton történik. Minden megrendelési eseményhez egyértelműen tartozik egy bizonylat, és fordítva, minden bizonylat egy rendelési esemény következménye. Nézőpont kérdése, hogy a modellezésnél a bizonylatról, mint fizikai objektumról mondunk állításokat, amiket utána egy relációban írunk le, vagy a megrendelésről, mint eseményről tesszük ugyanezt. (Természetesen nem mindig mindegy, hogy mit választunk, a választásnak következményei is lehetnek.) Ha a bizonylaton lévő információt mondatokban szeretnénk elmondani, akkor, amiről állítunk valamit, azokat hívjuk egyedeknek, amit állítunk ezekről, az az egyed egy tulajdonságának az értéke. A relációs adatmodellezés lényege, hogy ezek a mondatok minél egyszerűbbek és tömörebbek legyenek, ne tartalmazzanak ismétlődést, idegen szóval redundanciát. Utána az így kapott állításokat valamilyen egyszerű formalizmussal, számítógép segítségével feldolgozható táblázatokba rendezzük.

Formálisan a relációs adatmodell táblázatokból (relációkból) áll. Minden táblázatnak van neve, és ez a név egyedi az adatbázison belül. Egy táblázatnak meghatározott számú oszlopa és tetszőleges számú sora van. Az oszlopoknak a táblázaton belül egyedi nevének kell lennie. Egy oszlopban azonos típusú adatok lehetnek csak. Minden cellában csak egy érték lehet. A táblázatban nem lehet két azonos sor.

A relációs adatmodell nem foglalkozik a fizikai megvalósítással. Az, hogy a rendszer milyen technikákat alkalmaz a gyors visszakeresésre, hogyan tárolja a táblázatokat a háttértárolókon stb., az adatbázis fizikai tervezéséhez tartozik, erősen rendszerfüggő, ezért nem foglalkozunk vele. Az SQL tábladefiníciójában mondhatunk fizikai, tárolási információkat is, de ez túlmutat a példatár által felvállalt területen.

Tekintsük a bizonylatot és a rajta lévő adatokat:

RENDELLS					BÜTYKÖLŐ Kft 3333 Nagypusztza Almavirág út 26. Adószám: 11111111			
Megrendelő Név: Cím: Adószám:					Rendelés dátuma:		Rendelésszám: Ab-1234	
VTSZ	termékkód	termék név	mennyiség	nettó egységár	ÁFA kulcs	bruttó egységár	bruttó ár	vállalt határidő
					Összesen			
a megrendelő aláírása					a vállalkozó aláírása			

Első feladat, hogy kiválasszuk a bizonylatról azokat az adatokat, amelyekre a tervezett kérdések megválaszolásához szükségünk lesz. Alapszabály: előbb kell meghatározni, hogy mit akarunk elérni, és utána határozhatjuk meg, hogy a cél érdekében mire van szükségünk. Ez nem mindig nyilvánvaló! Csak annyi adatot, pontosabban adatfeleséget, más szóval adattípust, vagy tulajdonságot válasszunk ki, amennyire feltétlenül szükségünk van! A „gyűjtögetünk, majd jó lesz valamire” elv mindig sok pénzbe kerülő kudarcot fog eredményezni. A kiválasztott adatfeleségeket nevezzük el valamilyen névvel.

A kiválasztott adattípusok:

név, cím, adószám, rendelés dátuma, rendelésszám, termékkód, terméknév, mennyiség, bruttó egységár, vállalt határidő

Elnevezésük:

név, cím, adószám, rend dátum, rend szám, kód, név, darab, ár, dátum

A bizonylatról kiválasztott adattípusokat két csoportba sorolhatjuk, egy részük egy bizonylaton csak egyszer fordulhat elő (pl.: a megrendelés dátuma), más részük többször, fizikai korláttól eltekintve, akárhányszor (pl.: termékkód). Elméletben „daraboljuk fel ollóval” a bizonylatot olyan csíkokra, amelyek egy adattípusból már csak egy előfordulást tartalmaznak (a bizonylat minden tételesora legyen egy önálló papírcsík). A darabolással akkor nem veszítünk információt, ha az összes feldarabolt bizonylatot bármikor „össze tudjuk ragasztani” úgy, hogy az eredeti bizonylatokat kapjuk vissza. Azért, hogy az összeragasztás elvégezhető legyen, a csíkokat ki kell egészíteni azzal az információval, hogy melyik bizonylathoz tartozott. Minden bizonylatnak kell tehát lennie egy egyértelmű azonosításának. Mivel minden hivatalos bizonylat egyértelmű azonosíthatóságát a számviteli szabályok is előírják, a bizonylaton van egy azonosító, esetünkben ez a rendelésszám.

Vizsgáljuk most meg az így kiegészített papírcsíkokat, azaz a tételesorokat, hogy lehet-e közöttük két azonos sor. A feladat eredeti leírása szerint egy termék egy bizonylaton csak egyszer fordulhat elő, a rendelésszám és a termékkód tehát minden tételesoron különböző lesz. Ez nem magától értetődő, ez a feladat leírásából következik!

Ezek után kétféle adatcsoportunk maradt, azaz az adatokat két táblázatba csoportosítva írhatjuk fel. Nevezzük el adatcsoportokat a feladat megoldásához szükséges adatrendszeren (adatbázison) belül egy egyedi névvel.

Az első táblázatban azokat az adattípusokat soroljuk fel, amelyek egyszer fordulnak elő egy bizonylaton:

RENDELÉSFEJ				
név	cím	adószám	rend dátum	rend szám

A második táblázatban pedig azokat, amelyek többször is előfordulhatnak. Mint fentebb meg gondoltuk, a bizonylat sorait kiegészítettük a bizonylat egyértelmű azonosítójával, a rendelésszámmal:

RENDELÉS

rend szám	kód	név	darab	ár	dátum

Mindkettőben van egyértelmű azonosító, amelyik a táblázaton belül csak egyszer fordulhat elő. Természetesen maga az adatsor is csak egyszer fordulhat elő, de a legkisebb olyan részét keressük az adatcsoportnak, amelyik csak egyszer

fordulhat elő, ezt nevezzük az adatsor kulcsának (rend szám, illetve rend szám és kód). Egy táblázaton belül minden adattípusnak különböző neve van. A név elnevezés kétszer fordul elő, de két különböző táblában. A következőkben a táblákat (relációkat) így fogjuk felírni:

RENDELÉSFEJ{rend_szá, név, cím, adószám, rend_dátum}
RENDELÉS{rend_szá, kód, név, darab, ár, dátum}

Egy megrendelő több bizonylaton is előfordulhat. Vegyük ki magára a megrendelőre vonatkozó adatokat, és helyettesítsük azokat egyetlen kóddal, amit a továbbiakban hívunk partnerkódnak (partner_kód). Vegyük észre, hogy a bizonylaton a partnerkód nem szerepelt, ezt a fogalmat a modellezés során mi hoztuk létre. A rendszerben előforduló kódokat mi hozzuk létre azért, hogy a hivatkozásokat egyszerűbbé tegyük, védjük a rendszert az elgtelési hibáktól stb. Természetesen kifelé, a felhasználó felé ezeket a kódokat nem fogjuk megjeleníteni, de ez már a rendszertervezés későbbi feladata.

Ugyanígy hagyjuk el a tételesorokból magára a termékre jellemző adatokat, és csak a termék kódját hagyjuk meg, az elhagyott adatokból készítsünk külön táblázatot. A termékek elkészültét is tárolni szeretnénk, ezért a RENDELÉS táblába, a rendelt termékek mellé vegyünk fel még egy mezőt, amelyben majd azt fogjuk tárolni, hogy az adott termékből minden darab elkészült-e már vagy sem.

Az eredményt négy táblázatba foglalhatjuk össze, ahol a táblázatok sorainak az egyediségét biztosító mezőket, a kulcsokat aláhúzott betűkkel jelöltük.

PARTNER{partner_kód, név, cím, adószám}
RENDELÉSFEJ{rend_szá, partner_kód, rend_dátum}
RENDELÉS{rend_szá, kód, darab, dátum, kész} TERMÉK{kód,
név, ár}

A továbbiakban a PARTNER táblát nem fogjuk használni.

Ezeket az adattáblázatokat harmadik normálformájú relációknak hívjuk. (Van első, második, harmadik, sőt további normálformák is, de ezeket itt nem részletezzük, az olvasó több jegyzetben, könyvben megtalálhatja a matematikai szempontból is pontos, részletes leírásukat.)

Ezt a folyamatot, amikor a kiválasztott adatokból harmadik normálformájú relációkat állítunk elő, normalizálásnak hívjuk.

Harmadik normálformájú egy reláció (egy adattábla), ha:

- van a relációnak az adatbázison (az adattáblák csoportján) belül egy egyedi neve,
- minden reláció meghatározott számú tulajdonságot kapcsol össze (a táblának meghatározott számú oszlopa van), és minden sor egy konkrét egyed előfordulásról mondja el az adott sorrendben a tulajdonságok értékeit,
- a reláción belül minden tulajdonságnak (oszlopnak) egyedi neve van,

- egy sornak egy tetszőszerinti mezőjében csak egy érték szerepel (nem tartalmazhat listát, struktúrát stb.),
- a reláción belül nincs két egyforma sor,
(ebből következik, hogy a reláción belül van a mezőknek olyan legkisebb csoportja, amelyekben biztosan különbözik két sor, és ezt nevezzük a reláció kulcsának)
- nincs a relációnak a kulcsot kivéve más olyan mezőcsoportja, akár a kulcs részeitől választva ki azokat, akár tetszőszerinti más mezőkből, amelyeknek az értékét ismerve, további mezők értékét minden esetben biztosan meg tudjuk állapítani, azaz nem tartalmaz redundáns adatokat.

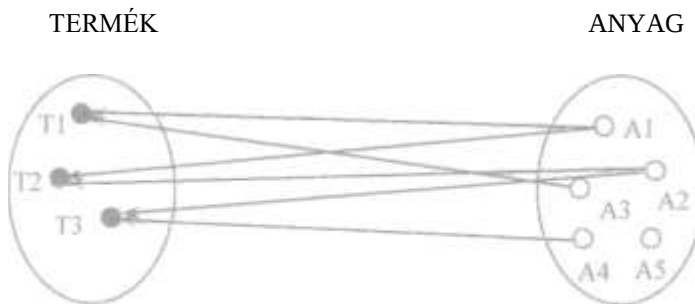
Pontosabb definícióhoz több matematikai formalizmus szükséges, de ettől itt eltekintünk.

II. problémakör

Egy termék többféle anyagból épül fel. Szeretnénk a termékek felépítését meghatározó információból is relációt, relációkat létrehozni.

A valóság

A termék kódja legyen T1, T2 ... , az anyagé pedig legyen A1, A2 ...



Egy termék többféle anyagból épül fel, egy anyag többféle termékbe épül be. Sem egy anyag nem határoz meg egyértelműen 1 terméket, sem egy termék nem határoz meg egyértelműen 1 anyagot. Másképpen kifejezve: sem egy anyagnak nem egyértékű tulajdonsága, hogy milyen termékbe épül be, sem a terméknek nem egyértékű tulajdonsága, hogy milyen anyagból épül fel.

A termékek és az anyagok között a valóságban $N : M$ kapcsolat van! Szokták ezt a kapcsolatot több - több kapcsolatnak is nevezni.

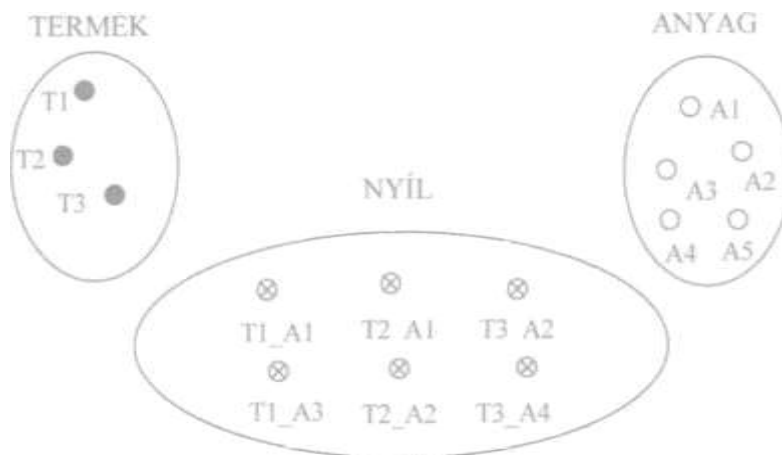
Az ilyen kapcsolatot ebben a formában a relációs adatbázis szemlélet nem tudja kezelni!

Modell_1

Tekintsük az előző ábrát! Az ábra 2 db ponthalmazból és 1 db nyilakat tartalmazó halmazból áll. Minden pontnak, függetlenül attól, hogy melyik halmazhoz tartozik, van egyértelmű neve, azonosítója.

A nyilaknak nincs nevük, de azonosítani tudjuk őket azzal, hogy honnan indultak ki, és hova érkeznek meg, vagy, ami ugyanúgy egyértelmű, hova érkezik és honnan indult ki. Legyen a nyilak azonosítója pl.: T1 A1, T1A3 stb.

Ábrázoljuk a három halmazt! A nyilak legyenek a NYÍL halmaz elemei.



Nézzük meg, mi az értelme ennek a NYÍL halmaz elemeinek.

A nyíl az eredeti ábrában azt fejezte ki, hogy „egy anyag milyen termékekbe épül be”, vagy fordítva, azt mondhattuk volna, hogy „egy termék milyen anyagokból épül fel”. Vegyük észre, hogy a két mondatban más az alany (anyag, illetve termék), és amit állítunk, az mindkét esetben többes számban van (termékek, anyagok). N:M kapcsolat van a termékek és az anyagok között.

A nyilak halmaza lényegében megállapítások halmaza, egy adott anyagról teszünk egy megállapítást, hogy egy adott termékkel olyan viszonyban van, hogy abba beépül. Ebben az állításban már nincs többes szám. Egy anyagról és egy termékről szól az állítás.

Fogalmazhatjuk úgy is az állítást, hogy egy nyíl az előző ábrában, vagy egy jel a mostani modellben egy eseményt jelöl, egy bizonyos anyag egy bizonyos termékbe beépítésre került. Egy ilyen állítás, illetve esemény önmagában nem határozza meg egy termék szerkezetét, csak ha minden az adott termékre vonatkozó állítást együttesen vesszük figyelembe, akkor tudjuk a termék szerkezetét meghatározni. Az egyes állításoknak önmagukban is igaznak kell lenniük, és ha az állításhalmaz teljes, azaz minden nyilat felvettünk az állítások közé, akkor mondhatjuk, hogy jó a modell.

Nevezzük a NYÍL halmazt a továbbiakban SZERKEZET halmaznak, hiszen a termékek szerkezetéről szól.

Vegyük észre, hogy ez a modell tartalmazza mindazt az információt, amit az előző ábrán felrajzoltunk.

Modell_2

A modell_1 elemei között kétféle egyértelmű (1 : N) kapcsolat van, más elnevezéssel kétféle egy a többhöz kapcsolat van.

A SZERKEZEI egy eleme egyértelműen meghatároz pontosan 1 anyagot (amit beépítettünk) és pontosan 1 terméket (amibe beépítettük). Fordítva: az ANYAG halmaz egy eleméhez tartozhat 0, 1 vagy több elem a SZERKEZET halmazból, és a TERMÉK halmaz egy eleméhez 1 vagy több elem tartozhat a SZERKEZET halmazból. A lényeg, hogy mindkét esetben lehet több is, egyik esetben sem határoz meg egyértelműen egy elemet.

Röviden úgy fogalmazhatjuk a SZERKEZET és az ANYAG kapcsolatát:

a SZERKEZET meghatározza az ANYAG-ot,

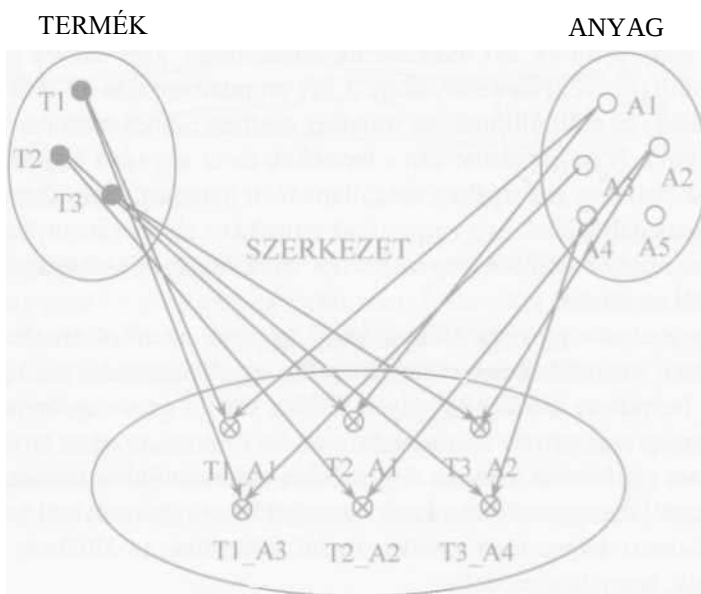
de az ANYAG nem határozza meg a SZERKEZET-et.

Természetesen, ha kiválasztunk egy anyagot, akkor megtalálhatjuk a hozzá tartozó szerkezet bejegyzéseket, de erre nem mondjuk azt, hogy meghatározza. A meghatározza fogalmat fenntartjuk az egyetlen érték meghatározására.

A kapcsolatok tehát:

TERMÉK : SZERKEZET 1 : N,

SZERKEZET : ANYAG N : 1.



Relációs adatmodell

A relációs adatbázismodell fizikailag csak táblázatokat tárol, a táblázatok közötti kapcsolatokat nem tárolja. Két tábla közötti kapcsolatot a táblákban lévő mezők értékei fejezik ki. A lekérdezésekben az egyik legtöbbször felhasznált kapcsolat a kulcs - külső kulcs kapcsolat. Egy táblában egy mező (mezőcsoport) akkor külső kulcs, ha ennek a mezőnek (mezőcsoportnak) az értékei egy másik tábla kulcs mezőjének (mezőcsoportjának) az értékei között megtalálhatóak, azaz a két mezőnek (mezőcsoportnak) ugyanaz az értékkészlete, és az egyik kulcs mező (mezőcsoport).

A terméknek a kód nevű tulajdonsága az, amelyik egyedi értékeket tartalmaz, azaz, amelynek az értéke a termék azonosítója: T1, T2, T3... Ez azt jelenti, hogy a termékek tulajdonságait leíró táblázatban a kulcs a kód mező.

Ugyanígy az anyagokat leíró tulajdonságok között az azonosító nevű tulajdonság tartalmaz egyedi értékeket: A1, A2, A3... Az anyagok tulajdonságait leíró táblázatban a kulcs az azonosító mező.

A SZERKEZET egy pontjának egyértékű tulajdonsága, hogy melyik termék és melyik anyag viszonyáról mond valamit, pl.: T1_A1 a TI termékről mondja azt, hogy van benne A1 anyag. Relációs szemléletre átfordítva ez azt jelenti, hogy a SZERKEZET táblának van két mezője, az egyik a termék kódjait (kód), a másik az anyag azonosítóit (azonosító) tartalmazza. Nevezzük ezeket is kód és azonosító mezőnek. A két mező együttesen egyértelműen meghatároz egy sort a táblában, azaz a kettő együttesen összetett kulcsa SZERKEZET táblának. A relációs adatbázisban egy mezőre a mező nevével hivatkozunk, de ha a mezőnév nem jelent egyértelmű hivatkozást, mert egy másik táblában is van ugyanilyen nevű mező, akkor a mező nevét egészítsük ki a tábla nevével (pl.: szerkezet.kód, ter- mék.kód), és így már egyértelmű hivatkozást kapunk.

A TERMÉK és a SZERKEZET táblák között a kapcsolatot a termék.kód és a szerkezet.kód mező értékei, mint kulcs - külső kulcs kapcsolat tartalmazza. Röviden úgy fogalmazhatjuk, hogy a kapcsolat hordozója a külső kulcs, a mezőértékek azonossága létesíti a konkrét kapcsolatot. Ugyanígy az ANYAG és a SZERKEZET táblák közötti kapcsolat, amelyet az anyag.azonosító és a szerkezet.azonosító kulcs - külső kulcs mező hordoz. A TERMÉK és az ANYAG között a relációs szemléletben nincs kapcsolat.

A relációs adatmodell tehát:

```
TERMÉK{kód, ...}
SZERKEZET{kód, azonosító, ...} ANYAG{azonosító, ...}
```

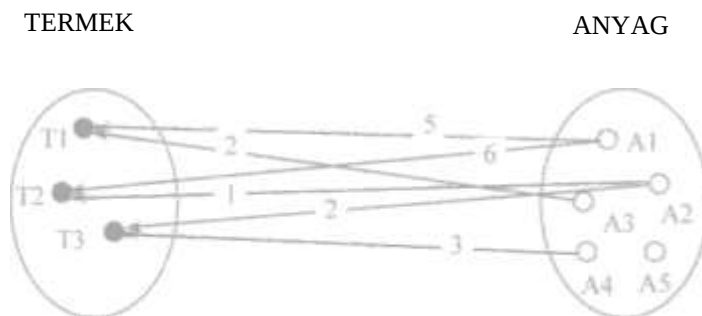
Vegyük észre, hogy bár az ábra teljesen szimmetrikus, az adatbázisban a TERMÉK és az ANYAG nem teljesen ugyanolyan tulajdonságokkal rendelkezik.

A SZERKEZET tábla kód mezőjére és az azonosító mezőjére is igaz, hogy csak olyan értékeket tartalmazhat, amelyek megvannak a TERMEK tábla kód mezőjében, illetve az ANYAG tábla azonosító mezőjében. A TERMÉK tábla kód mezője minden értékének legalább egyszer elő kell fordulnia a SZERKEZET tábla kód mezőjében, mert, a feladat megfogalmazásából következően, minden termék legalább egyféle anyagból áll. Az ANYAG táblának az azonosító mezőjére ilyen megszorítás nem igaz. Lehet, hogy előfordul egyszer vagy többször, de lehet, hogy egyszer sem. Lehetnek olyan anyagok a nyilvántartásban, amelyek a jelenlegi termékek egyikébe sem kerülnek beépítésre (pl.: elfekvő készlet).

A valóság pontosítása

A valóságot ábrázoló első ábrán a nyilaknak kétféle tulajdonságuk volt, honnan indultak ki, és hova érkeztek meg. A felhasználás szempontjából fontos információ az is, hogy hány egységnyi anyagot építünk be az adott termékbe. Minden kapcsolatot kifejező nyílhoz tartozik egy ilyen érték, azaz a nyilaknak van egy további tulajdonságuk is. Írjuk ezeket az értékeket a nyilakra. Az A1-et és a T1-et összekötő nyíl feletti 5-ös szám azt jelenti, hogy a T1 termékbe A1 anyagból öt egységet kell beépíteni. Az ábrából az is következik, mivel két pontot csak egy nyíl köthet össze, hogy nincs is több egységnyi A1 anyag a T1 termékben.

Az anyagok beépülését a termékekbe tehát így ábrázolhatjuk:



A relációs szemléletben ez úgy fogalmazható, hogy a szerkezet egyednek van egy olyan tulajdonsága is (mennyiség), amelyik a beépítendő anyag mennyiségét adja meg, más szavakkal a SZERKEZET táblának van egy olyan mezője is, amelyik a beépítendő anyag mennyiségét tartalmazza. A beépítésnek, mint eseménynek egyrészt van azonosításra használható két tulajdonsága, másrészt a beépítésről szóló mondatnak van egy további bővítménye is, pl.: a T1 termékbe az A1 anyagból 5 egységnyt kell beépíteni. (Többek között ezért is érdemes tisztázni, hogy miről szól egy tábla!)

Az anyagoknak a feladat szempontjából fontos további tulajdonsága a neve, mértékegysége, az egységára és a készlet.

A végleges relációs adatbázis

A kialakult és a példatárban a továbbiakban használt adatbázist a következő relációk írják le:

RENDELÉSFEJ{rend_száma, partner_kód, rend_dátum}

RENDELÉS{rend_száma, kód, darab, dátum, kész}

TERMÉK{kód, név, ár}

SZERKEZET{kód, azonosító, mennyiség}

ANYAG{azonosító, neve, mért_egys, egys_ár, készlet}

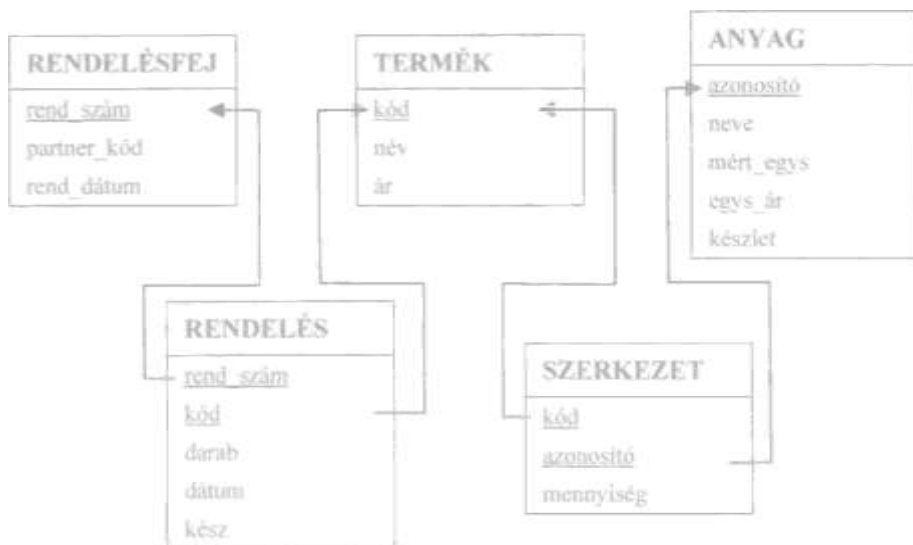
Az adatbázis bemutatása

Az adatbázis szerkezete

Az előző fejezetben meghatároztuk azokat a relációkat, amelyeket az adatbázisunkban tárolni szeretnénk:

RENDELÉSFEJ{rend_száma, partner_kód, rend_dátum}
RENDELÉS{rend_száma, kód, darab, dátum, kész}
TERMÉK{kód, név, ár}
SZERKEZET{kód, azonosító, mennyiség}
ANYAG{azonosító, neve, mért_egys, egys_ár, készlet}

A kulcs - külső kulcs kapcsolatok ábrázolására használható a táblázatok kulcs, illetve külső kulcs mezőit nyilakkal összekötő kapcsolati diagram. A nyíl a külső kulcstól mutat a kulcs felé.



Emlékeztetőül:

- **kulcs**: a tábla egy vagy több tulajdonságából álló csoport, amelyik egyértelműen meghatározza a tábla bármely sorának minden további értékét, és ha ebből a tulajdonságscsoportból elhagyunk egyetlen tulajdonságot, már nem lesz igaz az egyértelmű meghatározás, (következmény: értéke nem lehet üres, és benne minden érték csak egyszer fordulhat elő),

- *külső kulcs*: egy vagy több tulajdonságból álló csoport, amely egy másik táblának a kulcsa, (következmény: a kulcs és a külső kulcs értékkészlete azonos halmazból származik).

A definícióból következik, hogy mind a kulcs, mind a külső kulcs lehet egyszerű, egy mezőből álló, és lehet összetett, több mezőből álló. Az összetartozó kulcs, külső kulcspár szerkezetének természetesen azonosnak lennie. A külső kulcs mezői lehetnek kulcsszerű mezők vagy leírószerepű mezők (nem kulcsszerű mezők) a saját táblájukban.

Az előző diagramban egy nyíl úgy olvasható:

A TERMÉK táblában lévő kód mezőnek és a SZERKEZET táblában lévő kód mezőnek az értékkészlete azonos halmazból származik. A TERMÉK táblában a kód kulcs, ha tehát egy kódértéket kiválasztok, az a TERMÉK táblának egy sorát határozza meg (kettő nem lehet). Ugyanehhez a kódértékhez a SZERKEZET táblában több sor is tartozhat, mert SZERKEZET táblában a kód nem kulcs, az, hogy a kulcs része, ebből a szempontból lényegtelen. Következmény: egyrészt a kód mező a SZERKEZET táblában külső kulcs, másrészt a TERMÉK és a SZERKEZET tábla között 1 : N kapcsolat van.

Szokásos jelölése az adatbázisnak a következő ábrázolás is:

	rend szám		kód	
rend szám	kód	kód	azonosító	azonosító
RENDELÉSFEJ	RENDELÉS	TERMÉK	SZERKEZET	ANYAG
partner kód	darab	név	mennyiség	neve
rend dátum	dátum	ár		mért egys
	kész			egys ár
				készlet

A táblanevet vastag, nagybetűvel írjuk. A kulcsmező neveit a táblanév fölé, egyéb tulajdonságok neveit a táblanév alá írjuk.

Ebben a jelölésben a külső kulcsokat csak az azonos mezőnevek alapján lehetne felfedezni, márpedig a relációs adatbázis szemléletben a neveknek csak megkülönböztető szerepük van. A tulajdonságok értékkészletére vonatkozó információt csak az adatbázis leírását tartalmazó dokumentációból olvashatjuk ki. Természetesen célszerű valamilyen rövid emlékeztető nevet használni, de a mezőnévből nem lehet következtetni a külső kulcsokra. Az adatbázisnak ilyen ábrázolása esetén a kulcs - külső kulcs kapcsolatokat külön le kell írni.

Az adatok tárolása

A példatárban nem fogunk találkozni az adatbázisok fizikai tervezésével, az adatok tárolásával, azonban azért, hogy az olvasó a példatárban leírt feladatokat ki is tudja próbálni, az ACCESS adatbázis-kezelő rendszer lehetőségeit figyelembe véve megadunk egy lehetséges tárolási formát. Felhívjuk a figyelmet arra, hogy az itt leírt adattípusok nem felelnek meg a szabványos SQL adattípusoknak, ezek csak az ACCESS adatbázis-kezelő rendszerben érvényesek. Az SQL szabvány hasonló, de nem ugyanilyen. A tárolásnál előírt adattípusokat és az adattartalomra vonatkozó előírásokat az SQL lekérdezésekben fel is fogjuk használni. Egyszerűség kedvéért a fenti adatbázis leírást egészítettük ki a mezők tárolására vonatkozó információval.

	rend szám szöveg 7		kód szöveg 3	
rend_szá szöveg 7	kód szöveg 3	kód szöveg 3	azonosító egész	azonosító egész
RENDELÉSFEJ	RENDELÉS	TERMÉK	SZERKEZET	ANYAG
partner kod szöveg 3	darab egész	név szöveg 30	mennyiség pénznem	neve szöveg 20
renddatum dátum	dátum dátum	ár h. egész		mért egys szöveg 3
	kész igen/nem			egysár pénznem
				készlet pénznem

Megszorítások, szabályok

Az ACCESS az adattípus meghatározásán túl, mint minden korszerű adatbázis-kezelő rendszer, számos finomítási lehetőséget tartalmaz, amellyel pontosíthatjuk a tárolási struktúrát és a mezők megjelenését. Megadhatunk beviteli maszkokat, érvényességi szabályokat, indexeléseket, formátumokat stb.

Ebben a példatárban a táblák definiálásánál a következő táblázatban leírt szabályokat, megszorításokat alkalmaztuk (adattípus, hossz, formátum, beviteli maszk, érvényességi szabály). A részletekre nem térünk ki, bármelyik ACCESS kézikönyvben vagy az ACCESS súgójában megtalálhatóak.

mezőnév	szabály
ár	szám, hosszú egész formátum: rögzített, 2 tizedes jegy
azonosító	szám, egészszám érvényességi szabály: >9^ A.ND <1000 (értelmesebb lenne: 3 karakteres szöveg, amely 3 számjegyet tartalmaz, de a lekérdezési példák miatt választottuk a szám adattípusú tárolást)
dátum	dátum/idő, rövid dátum érvényességi szabály: >Date()
egysár	pénznem formátum: pénznem, 2 tizedes jegy
1 készlet	pénznem formátum: 0,00;-0,00[Kék] ;"nulla"[Piros] (a pénznem adattípus tárolása mindig 4 tizedes jegy!)
1 - kód	3 karakteres szöveg felépítése: beviteli maszk: 2 nagybetű + 1 számjegy >LL0;;
mennyiség	pénznem formátum: rögzített, 2 tizedes jegy >0 érvényességi szabály:
mért egys	3 karakteres szöveg érvényességi szabály IN("kg";"m";"db")
név	30 karakteres szöveg indexelt, nem lehet azonos (a lekérdezésben nem mindig használtuk ki!)
partnerkód	3 karakteres szöveg felépítése: beviteli maszk: 3 számjegy 000
rend dátum	dátum/idő, rövid dátum alapértelmezett érték: -Date()
rendszám	7 karakteres szöveg felépítése: 1 nagybetű + 1 kisbetű + + 4 számjegy beviteli maszk: (a "-"jelet is tárolni kell!) >L<L\ -0000;0;

Természetesen a fenti szabályok nem mutatják be az ACCESS minden adatdefiníciós lehetőségét, ezek csak önkényesen kiemelt példák. Ha az olvasó más, Korszerű környezetben építi fel az adatbázisát, annak a rendszernek a dokumentációjában biztosan talál a fent leírtakhoz funkcióiban hasonló szabályokat.

Kidolgozott feladatok

A kidolgozott példák első hat csoportjában csak egy táblát használó SQL lekérdezések vannak. A feladatok megoldásánál bemutatjuk, bár nem részletesen az ACCESS tervező rácsának a használatát is. Amint a bevezetésben írtuk, ez a könyv nem ACCESS tankönyv. A könyv feladata az SQL bemutatása, és csak azért szerepel benne az ACCESS tervező rácsa, hogy az olvasó könnyebben tudjon elindulni az SQL megismerésének útján.

Az 5. példacsoportban egyszerű feladatokon keresztül a példatárban használt SQL lekérdezés záradékait mutatjuk be.

Nagyon fontos a 7. és 8. példacsoport megértése, hiszen az adatbázisokat és az SQL nyelvet csak több tábla esetén érdemes használni.

A 9. példacsoportban ugyanarra a kérdésre hétféle megoldást láthatunk. Ebben a fejezetben bemutatjuk, hogy egy feladatot megoldásához az SQL milyen különböző lehetőségeket biztosít.

Az alapfeladatok közé tartozik még a 10. és 11. feladatcsoport, amely egy tábla mezőjének a maximális értékének a meghatározását mutatja be.

All. feladatcsoportban megvizsgáljuk, hogy mi a különbség egy mező értéke maximumának és egy csoportosított mező értéke csoportonkénti maximumának a meghatározása között

A 12. feladatcsoporttól már a teljes SQL lekérdezési lehetőségeket kihasználjuk. A feladatokat nem nehézségi sorrend szerint csoportosítottuk, hanem inkább módszer vagy téma szerint.

Fontos megérteni a 13. feladatcsoportot, amely két szempont szerinti csoportosítással foglalkozik.

A 17. feladatcsoport gyakran okoz fejtörést az SQL elsajátításának kezdeti szakaszában.

Kiemelésre érdemes a 19. feladatcsoport, amely a predikátumok használatát foglalja össze.

A 22. feladatcsoport, amikor egy rendezett tábla sorainak folyamatos sorszámozását mutatjuk be, az SQL nem nyilvánvaló lehetőségeire mutat példát.

A 27-től a 29. feladatcsoport összetettebb, nehezebb feladatokat tartalmaz.

Az utolsó, a 30. feladatcsoportnál általánosítottuk az eddig használt adatmo- dellt. A feladatcsoport nem azért került a sorozat végére, mert nehezebb feladatok vannak benne, hanem azért, mert bővült az adatbázis szerkezete.

A feladatok részletes leírása

1. ÉS feltétel

Írassuk ki azoknak az anyagoknak az azonosítóját, amelyek 2-vel kezdődnek!

2. VAGY feltétel

Készítsünk egy olyan listát, amely tartalmazza a „cm”-ben mért anyagok azonosítóját és a „m”-ben mért anyagok azonosítóját is!

3. A logikai műveletek sorrendje

Keressük meg azokat a 2000. január 1.-re és 2.-re vállalt megrendeléseket, amelyekben „A” betűvel kezdődő kódú terméket rendeltek meg! Írassuk ki a feltételnek megfelelő rendelések minden adatát a RENDELÉS táblából!

4. A feltétel tagadása

Készítsünk egy listát, amely tartalmazza azoknak a termékeknek a kódját, darabszámát és a megrendelésszámát, amelyeket a mai naptól szám tva egy héten belül kell elkészíteni és még nincsenek készen!

5. Egyszerű lekérdezések egy táblából

1. Véletlen szám (FROM nélküli SELECT utasítás)
2. A mértékegységek listája (DISTINCT szerepe)
3. A tábla sorainak a száma (COUNT használata)
4. A különböző mértékegységek száma (COUNT(DISTINCT ...)) használata)
5. Határozzuk meg, hogy egyes betűkkel hány anyag neve kezdődik! (GROUP BY használata)
6. Csak azokat a betűket írassuk ki, amelyekkel legalább két anyag neve kezdődik! (HAVIN G használata)
7. Számoljuk össze, hogy az anyagok között mennyiből áll rendelkezésre 15 egységnél kevesebb és mennyiből 15 egységnél több vagy éppen 15 egység! (ORDER BY használata)

6. Paraméteres lekérdezés

1. Keressük meg, hogy „A” betűvel kezdődő nevű termék van-e az adatbázisban!
2. Keressük meg, hogy előre nem definiált, a lekérdezés idején megadott karakter sorozattal kezdődő nevű termék van-e az adatbázisban!

7. Táblák összekapcsolása, direktszorzat

1. Határozzuk meg, hogy egy adott termék milyen anyagokból épül fel! A termékeknek a nevét és a bennük lévő anyagoknak az azonosítóját írassuk ki!
2. A termék nevét és a benne lévő anyagok nevét írjuk ki!

8. Beágyazott SELECT utasítás

1. Keressük meg azoknak a termékeknek a nevét, amelyeknek az előállításához kell „113”-as azonosítójú anyag!
2. A „113” azonosítójú anyagokat tartalmazó termékeknek nemcsak a nevét, hanem a teljes szerkezetét, azaz minden anyagának az azonosítóját is írassuk ki!
3. A „113” azonosítót tartalmazó termékek teljes szerkezetének a kiírásánál a termék nevét, a benne lévő anyag nevét, a belőle szükséges mennyiséget és a mértékegységét is írassuk ki!

9. Csoportok kiválogatása

Határozzuk meg, hogy melyik megrendelésekben szerepel a „BC1 ” és a „BC9” kódú termék is! (elég a megrendelés számát kiírni)

10. Maximumkeresés

1. Határozzuk meg, hogy hányféle anyagból állnak a termékek!
2. Keressük meg, hogy melyik termékek állnak legalább háromféle anyagból!
3. Határozzuk meg, hogy maximálisan hányféle anyagból állnak a termékek!
4. Keressük meg, hogy melyek azok a termékek, amelyek a legtöbbféle anyagból állnak! (Csak a termékkódot kell kiírni.)
5. A legtöbbféle anyagból álló termékeknek nemcsak a kódját, hanem a nevét is írassuk ki!
6. Írassuk ki a termék kódját és azt az információt, hogy az adott termékben lévő anyagfélések darabszáma hogyan viszonylik a legtöbb anyagféléseget tartalmazó termék anyagféléségének a darabszámahoz! (A termék bonyolultságáról szeretnénk információt kapni.)

11. Csoportonkénti maximum keresése

1. A 2001. évi megrendeléseknek rendelésenként számítsuk ki a rendelés teljes értékét, és az eredménytáblát egészítsük ki a megrendelés hónapjának a sorszámaival!
2. Válasszuk ki a 2001. év legnagyobb megrendeléseit!
3. Havonként csoportosítással, válasszuk ki 2001. évi megrendelésekből a havi legnagyobb értékű megrendeléseket!

12. Adatbázisbeli megszorítások ellenőrzése

1. Határozzuk meg, hogy a termékek hányféle anyagból épülnek fel! Írjuk ki a termék nevét, valamint azt, hogy a termék hányféle anyagból épül fel!
2. Keressük meg azokat a termékneveket, amelyek nem egyediek!
3. Írassuk ki TERMÉK táblából azokat a kódokat, amelyek nem találhatók meg a szerkezet táblában, azaz azokat, amelyeknek nem adtuk meg a felépítését!

13. Két szempont szerinti csoportosítás

Gyűjtsük ki az adatbázisból, hogy melyik partner milyen termékből mennyit rendelt meg!

14. Az eredménytáblát szűkítő feltételek

1. Gyűjtsük ki az adatbázisból, hogy melyik partner milyen termékből mennyit rendelt meg 2001. januárjában!
2. Szűkítsük tovább az eredménytáblát, hogy csak azok a termékek kerüljenek be az összesítésbe, amelyekből az adott megrendelő több mint 500 darabot rendelt meg 2001. januáriában!
3. Szűkítsük tovább az eredménytáblát, hogy csak azok a termékek kerüljenek be, amelyek kódja „A” betűvel kezdődik, és amelyekből az adott megrendelő több mint 500 darabot rendelt meg 2001. januárjában!
4. Szűkítsük még tovább az eredménytáblát, hogy csak azok a termékek kerüljenek be az összesítésbe, amelyekből összesen legalább 1000 darabot rendeltek meg 2001. januárjában, és amelyek kódja „A” betűvel kezdődik, és amelyekből egy adott megrendelő több mint 500 darabot rendelt meg 2001. januárjában!

15. N : M viszonyú táblák összekapcsolása

Határozzuk meg, hogy mennyi a 2001. november 10.-re vállalt megrendelések összes anyagszükséglete! (Anyagonként ki kell íratni a szükséges anyag azonosítóját, nevét, az anyagból szükséges összes mennyiséget és az anyag mértékegységét!)

16. Legyártható-e egy termék

1. Határozzuk meg, hogy az 'AA1' kódú termék legyártható-e, azaz a termék előállításához szükséges anyagmennyiség rendelkezésre áll-e!
2. Határozzuk meg, hogy egy termék legyártható-e, és ha igen, hány darab gyártható le! A termék kódját paraméterként adjuk meg a lekérdezés idején!
- 3 Minden termékre határozzuk meg, hogy legyártható-e vagy sem!

I", \dott időintervallumban megrendelt, illetve meg nem rendelt termékek

1. Határozzuk meg, hogy melyik terméket (kód, elnevezés) rendelt meg már valaki (legalább egyszer megrendelték)!
- _ Határozzuk meg, hogy melyik termékre (kód, elnevezés) nem történt megrendelés még soha (senki sem rendelte még meg)!
3. Határozzuk meg, hogy melyik termék (kód, elnevezés) nem készült el legalább egy éve!

ÉL Egy adott termékcsoporthoz megrendelt megrendelések listája

Keressük meg azokat a megrendeléseket, amelyekben megrendelték az összes ■ an terméket, amelyeknek a kódja egy KERES nevű táblában van!

19. Az ALL, ANY és EXISTS predikátumok összehasonlítása

1. Keressük ki azokat a megrendelőket, akinek minden megrendelése 10000.- Ft felett volt!
2. Keressük ki azokat a megrendelőket, akiknek van olyan megrendelésük, amelyik 10000.- Ft vagy az alatt volt!
3. Keressük ki azokat a megrendelőket, akiknek minden megrendelése kisebb vagy egyenlő, mint 10000.- Ft!

20. Külső feltételek szerinti csoportokba sorolás

Minősítsük a termékeket a beépített anyagok összértéke szerint!

21. Sávosan adott kedvezmény

1. Egy megrendelés összértékéből, annak a nagyságától függően, százalékos árengedményt adnak. Számítsuk ki minden megrendelés összértékét és az engedmény nagyságát!
2. Hagyjuk el a KEDVEZMÉNY táblából a felső határokat (KEDV tábla), majd készítsünk egy lekérdezést a KEDV táblából, amelyik minden alsó határhoz kiszámítja a felső határt, azaz előállítja a KEDVEZMÉNY táblát!

22. Az eredménytábla sorainak sorszámozása

Számítsuk ki az egyes termékek bonyolultsági szintjét, amit az előállításához szükséges anyagfajták számával adunk meg! A bonyolultsági szint szerint adjunk a termékeknek egy folyamatosan növekvő sorszámot!

23. Előre meghatározott sorú táblázat készítése

1. Készítsünk év közben az adott év megrendeléseinek az összegéről havi összesített listát! Az utolsó hónapra tegyünk egy megjegyzést, hogy ez a hónap még lezáratlan, még érkezhetnek megrendelések!
2. Készítsünk olyan listát, amelyik tartalmazza az adott év minden hónapját, és a megrendelések összesített értékét, függetlenül attól, hogy az adott hónapra érkezett-e megrendelés! Amennyiben az adott hónapra még nem érkezhetett be megrendelés, mert az a hónap még nem kezdődött meg, a rendelés összértéke legyen üres. Módosítsuk továbbá az előző listát úgy, hogy a befejezett hónapok mellé kerüljön kiírásra, hogy az a hónap már lezárt!

24. Számítások a SELECT utasítás mezőlistájában

1. A cég 10% kedvezményt ad az eredeti árból azoknak a megrendelőnek, akik az „A” betűvel kezdődő kódú termékekből két különböző terméket vásárolnak. További 10% kedvezményt kap az a vásárló, aki olyan 2 különböző terméket vásárol meg, amelyeknek az összértéke 10000 Ft felett van. Készítsünk egy olyan lekérdezést, amelyik minden lehetséges kedvezményt kiír.

vezményes termékpárról megadja az eredeti árat, a kedvezmény nagyságát és a kedvezményezett árat!

2. A cég 10% kedvezményt ad egy megrendelés végösszegéből azokra a megrendelésekre, amelyekben szerepel „A” betűvel kezdődő kódú termékekből legalább két különböző termék. A megrendelés végösszegéből további 10% kedvezményt kap az a megrendelő, akinek a megrendelésének a végösszege nagyobb, mint 10000 Ft. Ha nem vásárolt meg legalább kétféle „A” betűvel kezdődő kódú terméket, nem kap semmilyen kedvezményt. Készítsünk egy lekérdezést, amelyik rendelésszámonként megadja a megrendelés kedvezmények nélküli végösszegét, a kétféle kedvezmény aktuális nagyságát, a teljes kedvezmény értékét és a kedvezmények figyelembevételével kialakított összértéket!

25. Azonos anyagokból felépülő termékpárok kikeresése

Keressük meg azokat a termékeket (termékpárokat), amelyek azonos anyagokból épülnek fel!

Segédfeladatok:

- 1.1. Határozzuk meg, hogy egy adott termékbe beépülő valamelyik anyag milyen más termékbe épül még be!
- 1.2. Határozzuk meg, hogy két termék felépítésének hány közös anyaga van!
- 1.3. Számoljuk ki és tároljuk, hogy egy termék hányféle anyagból épül fel!
- 1.4. Számítsuk ki, hogy két termékben hány közös anyag van! A termékpárokból hagyjuk ki azokat a párokat, amelyekben a termék különböző darabszámú anyagból épülnek fel!
- 1.5. Jelenítsük meg azokat a termékpárokat, amelyeknek minden anyaguk közös!
- II. 1. Jelenítsük meg azokat a termékkód párokat, amelyek azonos darabszámú anyagfajtából épülnek fel!
- II.2. Válogassuk ki azokat a termékkód párokat, amelyek nemcsak azonos számú anyagfélésegekből épülnek fel, hanem azonos anyagokból épülnek fel!

26. Különböző szerkezetű sorokból álló lista

- ^{1.} Készítsünk egy listát, amely egy paraméterrel megadott termékről megjeleníti, hogy legyártható vagy sem! Írassuk ki, a termék kódját és nevét, valamint azt, hogy legyártható, vagy nem gyártható le!
- 2 Készítsünk egy listát, amely üres, ha egy paraméterrel megadott termék legyártható, és ha nem gyártható le, akkor a terméknek adja meg azokat az anyagait, amiknek a kis készlete miatt a terméket nem tudjuk legyártani!
- 3 Egyesítsük a két listát, azaz egy termékről először állapítsuk meg, hogy legyártható vagy sem, majd, ha nem gyártható le, listázzuk ki azokat az anyagokat, amelyekből a készlet kevesebb, mint amennyi a termék legyártásához szükséges!

27. Összefokozatos lista

1. Készítsünk egy listát, amely tartalmazza minden megrendelés rendelésszámát és a rendelés összértékét! A sorokban jelenítsük meg a megrendelés évét és hónapját!
2. Készítsünk egy listát, amely tartalmazza megrendelések összértékét havi bontásban! A sorokban jelenítsük meg a megrendelés évét és hónapját!
3. Készítsünk egy listát, amely tartalmazza évente a megrendelések teljes összértékét! A sorokban jelenítsük meg a megrendelés évét!
4. Készítsünk egy listát, amely tartalmazza a megrendelések teljes összértékét!
5. Az elkészült négy eredménytáblából, készítsünk egyetlen, összefokozatos listát!

28. Külső ciklusváltozó szerinti válogatás

Ellenőrizzük, hogy az adatbázisban minden Aa-val és Ab-vel kezdődő rendelésszám megvan-e, azaz létezik-e Aa-0000 és Ab-9999 közé eső minden rendelésszám. Jelenítsük meg a hiányzó számlaszámokat!

29. Folytonos és nem folytonos mezőértékekből a hiányzó értékek megkeresése

1. Folytonos rendelésszámokat feltételezve keressük meg a rendelésfej táblából a hiányzó rendelésszám-intervallumokat!
2. Nem szükségképpen folyamatos rendelésszámokat feltételezve, keressük meg a hiányzó rendelésszámokat, és határozzuk meg ezeknek a darabszámát!

30. Az árak változásainak követése

I. Modell

1. Határozzuk meg, hogy mennyi az „AA6” kódú termék mai ára!
2. Határozzuk meg, hogy mi volt az „AA6” kódú terméknek az ára, amikor értékesítették az „Xy-1234” rendelésszámú megrendelésben! Jelenítsük meg azt is, hogy mikor történt a megrendelés!

II Modell

1. Határozzuk meg, hogy mennyi az „AA6” kódú termék mai ára!
2. Határozzuk meg, hogy mi volt az „AA6” kódú termék ára 2002. január 10-én!
3. Készítsünk el a 2002. január 1.-én érvényben volt árlistát!
4. Számítsuk ki, hogy átlagosan hány százalékkal emelkedtek az árak az év elejéhez képest!
5. Számítsuk ki, hogy a tárgyévi bevétel mennyivel és hány százalékkal lett volna kevesebb, ha év közben nem emeltünk volna árakat!

III. Modell

1. Határozzuk meg, hogy mennyi az „AA6” kódú termék mai ára!
2. Határozzuk meg, hogy mi volt az „AA6” kódú termék ára 2002. január 10-én!

1. ÉS feltétel

írassuk ki azoknak az anyagoknak az azonosítót, amelyek 2-vel kezdődnek!

A könyvnek, mint a bevezetőben említettük, nem célja, hogy az ACCESS rejtelseibe beavassa az olvasót, azonban a legtöbb olvasó számára a legkönnyebben elérhető az ACCESS SQL felülete. Az ACCESS lekérdezéseknek három nézete van: tervező nézet, SQL nézet és adatlap nézet. A tervező nézet és az SQL nézet, egymással teljesen összhangban, a lekérdezés megfogalmazásában segíti a felhasználót, az adatlap nézet pedig az összeállított lekérdezés eredményét mutatja meg. Ebben a részben bemutatjuk, hogy hogyan használhatjuk egyszerűbb feladatok megoldásánál az ACCESS tervező rácsát. A hangsúly természetesen itt is az SQL nyelven lesz. Nem fogjuk sokat magyarázni a rács szerkezetét, az ACCESS-ben megtalálható súgóban mindenki részletes magyarázatot találhat. Reméljük, hogy az olvasó, ahogyan előrehalad a feladatok megoldásában, belátja, hogy már közepes bonyolultságú lekérdezéseknél is hatékonyabb egy SQL utasítás beírása, mint a rács használata. A tervező rácsot csak a példatár első néhány feladatánál fogjuk használni, de az olvasó, ha ACCESS környezetben oldja meg a feladatokat, mindig megtekintheti, hogy a beírt SQL utasításnak a tervező rács milyen kitöltése felel meg.

A termékek előállításához felhasználható anyagokról szóló információ az ANYAG táblában van. Az azonosító 3 számjegyből áll, tárolása egészszám, ezért átfogalmazhatjuk a kérdést úgy, hogy melyek azok az anyagok, amelyeknek az azonosítója nagyobb vagy egyenlő, mint 200 és kisebb vagy egyenlő, mint 299.

I. Megoldás

Az ACCESS lekérdezés tervezőrácsán az egy sorba írt feltételeket az ACCESS „és” művelettel kapcsolja össze. Csak egyszer akarjuk kilistázni az azonosítókat, ezért a második oszlopban a megjelenítést ki kell kapcsolni.



<

Mező: Tábla:	azonosító	azonosító
Rendezés:	anyag	anyaa
Megjelentés:		
Feltétel: vagy:	>200	<=299

A rács kitöltéséből az ACCESS a következő SQL utasítást generálja:

```
SELECT ANYAG.azonosító
FROM ANYAG
WHERE (((ANYAG.azonosító)>=200)
AND ((ANYAG.azonosító)<=299));
```

A felesleges zárójeleket és minősítéseket elhagyva:

```
SELECT azonosító
FROM anyag
WHERE azonosító >= 200 AND azonosító <= 299;
```

A FROM záradékba fel kell tüntetni minden olyan táblázatot, amelyiket fel akarunk használni a lekérdezésben akár azért, mert megjeleníteni akarunk egy mezőt, akár azért, mert feltételt adunk meg rá a lekérdezés szűkítésénél. Egyszerűbben fogalmazva, csak olyan mezőt használhatunk fel a SELECT utasításban, amely a FROM záradékban szereplő táblázatból származik.

A SELECT utasítás mezőlistájába kell beírni a megjelenítendő mezőt vagy mezőket, esetünkben az azonosítót.

A WHERE záradékban szereplő logikai feltételt a táblázat minden sorára ki kell értékelni, és csak abból a sorból kell kiírni a SELECT mezőlistájában szereplő mezők értékét, amelyre a logikai feltétel igaz.

II. Megoldás

írjuk be az SQL nézetbe a lekérdezést a következő formában:

```
SELECT azonosító
FROM anyag
WHERE 200 <= azonosító AND azonosító <= 299;
```

A tervező nézetben ezt fogjuk látni:

Mező:	azonosító	200
Tábla:	anyag	anyag
Rendezés:		
Megjelenítés:	☒	☐
Feltétel:	<=299	<=[azonosító]
vagy:		

Az ACCESS létrehozott egy oszlopot, ami minden sorban azonos konstans értéket tartalmaz, és ezt hasonlítja össze az ANYAG tábla azonosító mezőjével. Ha a megjelenítést bekapcsoljuk, láthatjuk, hogy ebben az oszlopban mindig a 200, mint egészszám, jelenik meg. Az oszlop neve, mivel nem adtunk meg semmit, a használt ACCESS verziótól függően „Kifl” vagy „Expr100!” lesz. Ha a rácson akadjuk beírni ezt a megoldást, a mezőnevet a feltétel sorban szögletes zárójelbe kell tenni.

III. Megoldás

Megtehetjük, hogy a tervező rácson a feltétel sorba egy kifejezést írunk. Az AND operátorral összekapcsolt összetett feltételnél mindkét relációban a bal oldali mezőnév elmaradhat. Ha akarjuk, be is írhatjuk a mezőnevet, de akkor itt is szögletes zárójelbe kell tenni mindkétyszer a mezőnevet.

Mező:	azonosító
Tábla:	anyag
Rendezés:	
Megjelenítés:	☒
Feltétel:	>200 And <=299
vagy:	

Mező:	azonosító
Tábla:	anyag
Rendezés:	
Megjelenítés:	☒
Feltétel:	[azonosító]>200 And [azonosító]<=299
vaav:	

Az ACCESS ebből a rácsból a következő SQL utasítást generálja:

```
SELECT anyag.azonosító
FROM anyag
WHERE (([azonosító]>=200 AND [azonosító]<=299));
```

IV. Megoldás

A feladat a legegyszerűbben az SQL szabvány BETWEEN...AND művelete segítségével oldható meg.

A „kifejezés BETWEEN értékl AND érték2” művelet értéke igaz, ha a kifejezés értéke az értékl és az érték2 közé esik (a szélső értékeket is megengedve). Ellenkező esetben a művelet eredménye hamis.

Mező:	azonosító
Tábla:	anyag
Rendezés:	
Megjelenítés:	☒
Feltétel:	Between 200 And 299
vagy:	

A megoldás SQL formája:

```
SELECT azonosító
FROM anyag
WHERE azonosító BETWEEN 200 AND 299;
```

akorlő feladatok

1. Melyek a 2001. január 1. és január 10. közötti rendelések?
2. Melyek azok az anyagok, amelyeknek a neve „a” betűvel kezdődik és 1000 Ft- nál nagyobb az egységára?
3. Melyek azok az anyagok, amelyeknek a készlete egyenlő nullával, és a mértékegysége „kg”?

2. VAGY feltétel

Készítsünk egy olyan listát, amely tartalmazza a „cm”-ben mért anyagok azonosítóját és a „m”-ben mért anyagok azonosítóját is!

A biztosan rossz SQL megoldás

A „cm” és a „m” karakterkonstans, ezért idézőjelek közé kell tenni.

```
SELECT azonosító  
FROM anyag  
WHERE mért_egys = "cm" AND mért_egys = "m";
```

A tetszésszerű anyag mértékegysége nem lehet egyszerre egyenlő „cm”-rel is és „m”-rel is. A fenti lekérdezés biztosan üres listát ad vissza eredményként.

Bár a feladat megfogalmazása mindenki számára egyértelmű, fogalmazzuk át részletesebben a feladatot. Olyan listát kell készíteni, amelyik tartalmazza egyrészt a „cm”-ben mért, másrészt a „m”-ben mért anyagok azonosítóját. A lista tehát két részből tevődik össze, az egyik részben lévő sorokra az jellemző, hogy olyan anyagok azonosítóját tartalmazza, amelyeket „cm”-ben mérnek, és a másik részben a „m”-ben mért termékek azonosítója van. A megfogalmazás tehát listáról szól, amelyik kétfajta részből áll, és mindkettőt ki kell írni.

Másképpen fogalmazva, az eredménylista egy sorára azt mondhatjuk, hogy vagy az első részhez tartozik, vagy a második részhez. A lista tehát olyan soroknak az összessége, amely sorokra vagy az igaz, hogy „cm”-ben mért, vagy, hogy „m”-ben mért anyag azonosítóját tartalmazza.

A két megfogalmazás között az a különbség, hogy az első az eredménylistáról beszél (ezért szerepel benne és kötőszó), a második megfogalmazás egy sorról szól (ezért használtuk a vagy kötőszót).

Az SQL mindig sorokra vonatkozó állítást tartalmaz. Egy SQL utasítással olyan sorokat választunk ki a FROM záradékban megadott táblázatból, amelyek megfelelnek a WHERE záradékban megadott feltételnek.

Az eredeti kérdésnek megfelelő, átfogalmazott feladat tehát: készítsen egy olyan listát, amelynek egy sora olyan terméknek azonosítóját tartalmazza, amely terméket vagy „cm”-ben, vagy „m”-ben mérnek!

I. Megoldás

A tervező rácson egymás alatti sorokba írt feltételeket az ACCESS vagy logikai művelettel kapcsolja össze, a rácsot tehát a következőképpen tölthetjük ki:

Mező:	azonosító	mért egys
Tábla:	anyag	anyag
Rendezés:		
Megjelenítés:	☒	☐
Feltétel:		'cm'
vagy:		'm'

A SQL nyelv a vagy műveleti jelet természetesen az angol OR szóval jelöli:

```
SELECT ANYAG.azonosító
FROM ANYAG
WHERE (((ANYAG.mért_egys)='cm')) OR
      (((ANYAG.mért_egys)='m'));
```

A felesleges minősítések és zárójelek elhagyásával megkaphatjuk azt a formát, ahogy a lekérdezést legegyszerűbb formában a lekérdezés SQL nézetébe beírhatjuk.

```
SELECT azonosító
FROM anyag
WHERE mért_egys = 'cm' OR mért_egys = 'm';
```

II. Megoldás

Ebben az esetben is felcserélhetjük az egyenlőségjel két oldalán álló kifejezést. Cseréljük meg például a második feltételnél.

```
SELECT azonosító
FROM anyag
WHERE mért_egys = 'cm' OR 'm' = mért_egys;
```

A beírt SQL utasítást a rács így jeleníti meg:

Mező:	azonosító	mért egys	'm'
Tábla:	anyag	anyag	
Rendezés:			
Megjelenítés:	☒	☐	☐
Feltétel:		'cm'	
vagy:			[mért_egys]

Vegyük észre, hogy ez két különböző mezőre mond feltételt, a mért egys mezőre, és az 'm' konstansra. A két OR művelettel összekapcsolt feltételt a rácson két sorba kell írni. A harmadik oszlop feltétel sorában a mezőnevet szögletes zárójelek közé kell írni, hogy az ACCESS rács a karaktersorozatot mezőnévként tudja értelmezni, különben az ACCESS automatikusan idézőjelek közé tenné és konstansként értelmezné.

2. VAGY feltétel

III. Megoldás

A rács feltétel sorába közvetlenül is beírhatjuk az OR műveletet.

A konstansoknál az ACCESS alapértelmezés szerint az SQL szabvány szerinti aposztrófot használja, de elfogadja az idézőjelet is (nem a nyomdai idézőjelet). A beírásnál vegyesen használtuk a kétféle jelölést, így is elfogadja.

Mező:	azonosító	mért egys
Tábla:	anyag	anyag
Rendezés:		
Megjelenítés:	☒	☐
Feltétel:		'cm' Or 'm'
vagy:		

IV. Megoldás

Fogalmazhatjuk úgy is a kérdést, hogy egy azonosító akkor felel meg a lekérdezésnek, ha a saját sorában lévő mért egys mező értéke benne van a 'cm' és a 'm' konstansokból létrehozott halmazban. Az SQL utasítást legegyszerűbben a szabványos IN operátorral fogalmazhatjuk meg.

Mező:	azonosító	mért egys
Tábla:	anyag	anyag
Rendezés:		
Megjelenítés:	☒	☐
Feltétel:		In
vagy:		

SQL nézetben ezt így fogalmazhatjuk:

```
SELECT azonosító
FROM anyag
WHERE mért_egys IN('cm','m');
```

Vegyük észre, hogy a magyar területi beállításnak megfelelően a lista elválasztójele a rácson a pontosvessző, de az SQL nézetben marad a szabványos elválasztójel, a vessző.

Ha több, azonos mezőnévre vonatkozó, „vagy” műveleti jellel összekapcsolt feltételt kell megadnunk, az IN használata mindenképpen egyszerűbb, mint ha külön felsoroljuk a feltételeket és OR művelettel kapcsoljuk össze azokat.

Gyakorló feladatok

1. Melyik termékeknek a kódja kezdődik „AA” vagy „AB” betűsorozattal?
2. Melyik partnereknek a kódja végződik páratlan számmal?
3. Milyen kódú termékből rendeltek meg egy megrendelésben egy vagy két darabot?

3. A logikai műveletek sorrendje

Keressük meg azokat a 2000. január 1.-re és 2.-re vállalt megrendeléseket, amelyekben „A” betűvel kezdődő kódú terméket rendeltek meg! írassuk ki a feltételnek megfelelő rendelések minden adatát a RENDELÉS táblából!

Az SQL nyelvnek nincs olyan függvénye, amely egy mező értékének első karakterét adja vissza, pontosabban, egyetlen olyan függvénye sincs, amely egyet- .en mező értékéből egy értéket ad vissza, azonban az SQL utasításokban használhatjuk a befogadó nyelv függvényeit. Tekintettel arra, hogy a példatárban az iCCESS adatbázis-kezelővel mutatjuk meg az SQL használatát, a megoldáshoz használjuk fel az ACCESS Left(kód,l) függvényét, amely a kód mezőben lévő karaktersorozatból balról egy karaktert ad vissza.

I. Ami nem jó

A következő lekérdezés szintaktikailag jó. A rendelés.* a rendelés tábla összes oszlopát adja vissza.

```
SELECT rendelés.*
FROM rendelés
WHERE dátum = #01/01/2000#
OR dátum = #01/02/2000#
AND Left(kód,1)='A';
```

..Az eredmény azonban nem felel meg a feltételnek. Ez a lekérdezés visszaadja az összes 2000. január 1.-re vállalt megrendelést és ezeken kívül azokat a 2000. január 2. -re vállalt termékeknek a megrendelését, amelyeknek kódja „A” betűvel kezdődik.



<

Mező:	rendelés.*	dátum	Left([kód],1)
Tábla:	rei Jéles	rendelés	
Rendezés:			
Megjelenítés:	0		
Feltétel:		#2000.01.01 # W	
vagy:		#2000.01.02.#	

3. A logikai műveletek sorrendje

A logikai műveletek sorrendje az SQL lekérdezésnél is a matematikában megszokott sorrend. Az AND operátor erősebb, mint az OR operátor, ezért először hajtódik végre az AND, utána az OR operátor.

Vegyük észre, hogy az ACCESS automatikusan szögletes zárójelbe tette a függvény paramétereként használt mezőnevet. Nem szükséges így Imi, de az ACCESS átalakítja, ha nem így írtuk.

II. Ez már a jó megoldás

Mindkét dátumra igaznak kell lennie, hogy a kód „A” betűvel kezdődik, és mivel a két feltétel a vagy művelettel összekapcsolódik össze, mindkét sorba be kell írni a „A” konstans. A jó megoldáshoz így kell kitölteni a rácsot:

Mező:	rendelés.*	dátum	Left([kód];1)
Tábla:	rendelés	rendelés	
Rendezés:			
Megjelenítés:	0	□	□
Feltétel:		#2000.01.01.#	'A'
vagy:		#2000.01.02.#	'A'

A kitöltött rácsból az ACCESS a következő SQL utasítást generálja:

```
SELECT rendelés.*
FROM rendelés
WHERE (((rendelés.dátum)=#1/1/2000#)
      AND ((Left([kód],1))='A'))
      OR (((rendelés.dátum)=#1/2/2000#) AND
      ((Left([kód],1))='A'));
```

Megtisztítva a felesleges minősítésektől és zárójelektől így írhattuk volna a lekérdezést közvetlenül az SQL nézetbe:

```
SELECT rendelés.*
FROM rendelés
WHERE dátum=#1/1/2000# AND Left(kód,1)='A'
      OR dátum=#1/2/2000# AND Left(kód,1)='A';
```

vagy egyszerűbben, a zárójelekkel átírhatjuk a műveleti sorrendet:

```
SELECT rendelés.*
FROM rendelés
WHERE (dátum=#1/1/2000# OR dátum=#1/2/2000#)
      AND Left(kód,1)='A';
```

Az utasítás a rácson így fog megjelenni:

Mező:	rendelés.*	dátum	Left([kód];l)
Tábla:	rendelés	rendelés	
Rendezés:			
Megjelenítés:	0		
Feltétel:	#2000.01.01.# Or #2000.01.02.#		'A'
vagy:			

Vegyük észre, hogy a dátumot a rácson a Windows területi beállításának megfelelően jeleníti meg. Az ACCESS a rácson elfogad más olyan értéket is, amit dátumként tud értelmezni, de átírja a területi beállításnak megfelelően. Az SQL nyelvire viszont nem érvényes a területi beállítás! Az ACCESS a dátumot mind a rácson, mind az SQL nézetben az SQL szabványtól eltérően kettős kereszt (#) jelek közé teszi.

Ha a SELECT utasítás mezőlistája helyett minősítés nélkül használjuk a * karaktert, továbbiakban így fogjuk használni, az ACCESS a rácson elhagyja a fen- ti ábrán látható első oszlopot, és a lekérdezés tulajdonságai közé írja be, hogy minden mezőt a kimenetre kell küldeni.

Gyakorló feladatok

1. Melyek azok az anyagok, amelyek mértékegysége „kg” vagy „db”, és a készlete egyenlő nullával?
2. Melyek azok a termékek, amelyeknek az ára 10000 és 20000 Ft között van, és a nevük „o” vagy „ö” betűvel kezdődik?
3. Melyek azok a megrendelések, amelyeket 2001. január 2. és január 3.-án adtak

4. A feltétel tagadása

Készítsünk egy listát, amely tartalmazza azoknak a termékeknek a kódját, darabszámát és a megrendelésszámát, amelyeket a mai naptól számítva egy héten belül kell elkészíteni és még nincsenek készen!

A kész mező kétféle logikai értéket tartalmazhat, a logikai TRUE (igaz) értéket, ha a termék minden megrendelt darabja elkészült, és a logikai FALSE (hamis) értéket, ha nincs még kész minden darab. A feladat értelmében az adatbázisban nem tároljuk azt, hogy hány darab készült már el, és hány darabot kell még elkészíteni az adott megrendelés teljesítéséhez. Azt tehát, hogy még nincs készen, értsük úgy, hogy nincs még készen minden megrendelt darab.

I. Megoldás

A még el nem készült rendelésekre az jellemző, hogy a kész mező értéke egyenlő a logikai hamis értékkel, a feltételnek megfelelő sorokat tehát a következő SQL utasítás adja meg:

```
SELECT *  
FROM rendelés  
WHERE kész = FALSE;
```

Az ACCESS Date() függvénye a rendszer dátumát adja vissza. Paramétere nincs. Ha egy ACCESS függvénynek nincs paramétere, akkor üres zárójellel kell megkülönböztetni egyéb nevektől.

Ha csak azokat a rendeléseket kérjük, amelyeket a mai naptól számított egy héten belül kell elkészíteni, és még nincsenek készen:

```
SELECT *  
FROM rendelés  
WHERE kész = FALSE  
AND Date() <= dátum AND dátum <= Date() + 7;
```

Nem akarunk minden mezőt kiíratni. A SELECT utasítás mezőlistájában csak a kiíratandó mezőket felsorolva kapjuk meg a feladatnak megfelelő lekérdezés*

```
SELECT rend_szám, kód, darab, dátum  
FROM rendelés  
WHERE kész = FALSE  
AND DateQ <= dátum AND dátum <= DateQ + 7;
```

A lekérdezésnek megfelelő rács:

Mező:	rend szám	kód	darab	dátum	kész	Date()
Tábla:	rendelés	rendelés	rendelés	rendelés	rendelés	
Rendezés:						
Megjelenítés:	0	0	0	0	☐	☐
Feltétel:				<=Date()+7	Hamis	<=[dátum]
vagy:						

A Hamis logikai érték helyett a rácson írhatunk FALSE értéket is, de a Windows magyar területi beállítása miatt az ACCESS átírja Hamis értékre.

II. Megoldás

A fenti SQL utasítást írhatjuk kicsit egyszerűbben is:

```
SELECT rend_száma, kód, darab, dátum
FROM rendelés WHERE NOT kész
AND dátum BETWEEN Date() AND Date() + 7;
```

A kész mező értéke önmagában is Igaz vagy Hamis értékű, ezért az WHERE záradékban magát a mezőt is tagadhatjuk a NOT művelettel.

Az utasítás tervező nézetben:

Mező:	rend szám	kód	darab	dátum	kész
Tábla:	rendelés	rendelés	rendelés	rendelés	rendelés
Rendezés:					
Megjelenítés:	0	0	0	0	☐
Feltétel:	Between Date() And DateO+7				Hamis
vagy:					

Mint láthatjuk, a rácson a kész mező tagadása ugyanúgy jelenik meg, mintha a kész mező értékét hasonlítanánk össze a Hamis értékkel.

Gyakorló feladatok

1. Melyek a 2001. január 1. és január 10. közötti rendelések, amelyeket nem egyessel kezdődő partnerkódú megrendelő adott le?
2. Melyek azok az anyagok, amelyeknek a neve nem „csavar” és nem „alátét”, de a mértékegysége „db”?
3. Melyek azok a termékek, amelyeknek a kódja nem „AB” vagy „AC” karakterekkel kezdődik és az ára kisebb, mint 10000 Ft?

5. Egyszerű lekérdezések egy táblából

1. Véletlen szám (FROM nélküli SELECT utasítás)

A legegyszerűbb szabványos SQL utasítás egy SELECT parancsból és egy FROM záradékból áll. Előfordulhat, hogy a SELECT parancs nem hivatkozik semmilyen táblázatra. A különböző SQL megvalósítások ilyenkor egy fiktív táblanevet használnak. Az ACCESS egyszerűen elhagyja magát a FROM záradékot. A következő példa ezért nem tekinthető szabványos SQL utasításnak, de az ACCESS megengedi.

```
SELECT Rnd() AS véletlen,  
       Rnd() AS véletlen_más, Rnd(O) AS véletlen_ua,  
       Int(100*Rnd(0)+1) AS egy_100;
```

A SELECT utasítás négy véletlen számot ad vissza. Az első három értéke nulla és egy közötti tizedesszám, a nullát megengedve, az egyet nem. A második és harmadik szám értéke egyenlő. A negyedik értéke egy és száz közötti véletlen egészszám, a határokat is megengedve. A FROM záradék nélküli SELECT utasítás mezőlistájában beágyazott SELECT utasítás nem lehet. Az ilyen SELECT utasítás lényegében csak egyszerű, esetleg paraméteres, számolásokra alkalmas.

2. A mértékegységek listája (DISTINCT szerepe)

Az ANYAG táblából kérjük a mért egys mező kilistázását.

```
SELECT mért_egys  
FROM anyag;
```

Ez a SELECT utasítás nem tekinthető a feladat megoldásának. Ennek az eredménytáblának ugyanannyi sora lesz, mint az eredeti táblának, egy mértékegység többször fog szerepelni a listában. A mért egys nem kulcs az ANYAG táblában, ezért nincs rá garancia, hogy a mezőben előforduló értékek különbözőek. Ha csak a különböző sorokat akarjuk megjeleníteni, szükséges a DISTINCT használata.

```
SELECT DISTINCT mért_egys  
FROM anyag;
```

Lgyanezt az eredményt megkaphatjuk úgy is, hogy a mértékegységekből csoportokat képezünk, és csak a csoportosító mezőt írjuk ki.

```
SELECT mért egys  
FROM anyag  
GROUP BY mért_egys;
```

Ez a megoldás feleslegesen bonyolult, a megvalósítása nagyon erőforrás-igényes!

3. A tábla sorainak a száma (COUNT használata)

Az ANYAG tábla sorainak a számát megkapjuk a COUNT segítségével.

```
SELECT COUNT(*)  
FROM anyag;
```

Ha a * helyett egy mezőnek a nevét használjuk a COUNT aggregáló függvényben, akkor azoknak a soroknak a számát kapjuk meg, ahol a mező ki van töltve.

```
SELECT COUNT(mért_egys) FROM anyag;
```

4. A különböző mértékegységek száma (COUNT(DISTINCT ...) használata)

A táblában található különböző mértékegységek számát a COUNT és DISTINCT együttes használatával kaphatjuk meg.

```
SELECT COUNT(DISTINCT mért_egys) FROM anyag;
```

Megjegyzések:

1. Bár ez a lekérdezés az SQL szabvány szerint szintaktikailag helyes, sem az ACCESS 98, sem ACCESS 2000 nem fogadja el. Az ANSI-92 ,SQL lekérdezési módban már ACCESS-ben is használható ez a lekérdezés.

Az ACCESS környezetben a különböző mértékegységeknek a számát két lekérdezés használatával kaphatjuk meg:

```
SELECT DISTINCT mért_egys  
FROM anyag;
```

KÜLÖNBÖZŐ{mért_egys}

```
SELECT COUNT(*)  
FROM különböző;
```

2. A következő két lekérdezés azonos eredményt ad:

```
SELECT COUNT(mért_egys)  
FROM anyag;
```

```
SELECT DISTINCT COUNT(mért_egys)  
FROM anyag;
```

Az első lekérdezés eredménye egyetlen sor, és a másodikban azt írtuk elő, hogy ezen egyetlen sorból csak a különbözőket jelenítse meg, ez pedig az előző lekérdezésben megkapott egyetlen sor.

5. Határozzuk meg, hogy egyes betűkkel hány anyag neve kezdődik! (GROUP BY használata)

Először határozzuk meg, hogy hány anyag neve kezdődik 'a' betűvel.

```
SELECT COUNT(*)  
FROM anyag  
WHERE Left(neve,1) = 'a';
```

Ha nemcsak az 'a' betűvel kezdődőek számára, hanem mindenféle betűvel kezdődőekre vagyunk kíváncsiak, egészítsük ki a táblát egy új mezővel, ami a név első betűjét tartalmazza.

```
SELECT *, Left(neve,1) AS első  
FROM anyag;  
-> ELSŐ_BETŰ{azonosító, neve, mért_egys, egys_ár, készlet, első}
```

Készítsünk csoportokat az első nevű mező alapján, és határozzuk meg, hogy az egyes csoportokba hány sor tartozik.

```
SELECT első, COUNT(*) AS darab  
FROM első_betű GROUP BY első;
```

Azokhoz a betűkhöz, amelyekkel nem kezdődik egyetlen név sem, nem fog tartozni egyetlen sor sem, és így nem készül róla eredmény sor sem. Nem az ábécé betűihez készít csoportokat, hanem az első nevű mező létező különböző értékeihez készít egy-egy csoportot, és minden csoportból készít egy sort.

A két lekérdezést egy lekérdezéssel is helyettesíthetjük, ha a GROUP BY záradékban a mező helyén bármilyen kifejezés is szerepelhet.

```
SELECT Left(neve,1) AS első, COUNT(*) AS darab FROM anyag  
GROUP BY Left(neve,1);
```

Megjegyzés:

Kulcs mezőre történő csoportosítást az SQL szintaktikája nem tiltja, de nincs sok értelme.

A következő két lekérdezés pontosan ugyanazt az eredményt adja:

```
SELECT azonosító, neve, egys_ár  
FROM anyag;
```

```
SELECT azonosító, MAX(neve), SUM(egys_ár)  
FROM anyag  
GROUP BY azonosító;
```

Az azonosító nevű mező az ANYAG táblában kulcs, annyi különböző értéke van, ahány sora van az ANYAG táblának. Ha az azonosító különböző értékeire csoportosítunk, annyi csoportot kapunk, ahány sora van a táblázatnak. Minden csoportban pontosan egy sor lesz. Egy csoportban az ábécé szerint legnagyobb név azonos a csoport egyetlen sorában lévő névvel, és a csoportban lévő sorok egységárainak az összege egyenlő a csoportban lévő egyetlen sorhoz tartozó egységárral.

6. Csak azokat a betűket írassuk ki, amelyekkel legalább két anyag neve kezdődik! (HAVING használata)

A csoportok közül csak azokat kell kiírni, amelyeknél a csoportra jellemző érték, a csoportban lévő sorok száma nagyobb vagy egyenlő, mint kettő. A WHERE záradék segítségével a feladatot nem tudjuk megoldani, A WHERE záradékban lévő feltételt a FROM záradékban lévő tábla soraira kell kiértékelni, egy sorból pedig nem állapítható meg, hogy hány vele egy csoportban lévő sor van még. A GROUP BY záradékkal csoportosított táblázatból válogat a HAVING záradék feltétele.

```
SELECT Left(neve,1) AS első  
FROM anyag  
GROUP BY Left(neve,1) HAVING COUNT(*) >= 2;
```

Mint látható, olyan kifejezésre adtunk meg feltételt, amely nem szerepelt a SELECT mezőlistájában, de szerepelhetett volna.

7. Számoljuk össze, hogy az anyagok között mennyiből áll rendelkezésre 15 egységnél kevesebb és mennyiből 15 egységnél több vagy éppen 15 egység! (ORDER BY használata)

A GROUP BY záradékba logikai kifejezés is írható, amelyik a sorokat két csoportra osztja aszerint, hogy a kifejezés igaz a sorra vagy sem.

```
SELECT Countf)  
FROM anyag  
GROUP BY készlet <15;
```

Az eredménytáblának a várakozásunkkal ellentétben 3 sora lesz, mert azokat a sorokat, amelyekben a készlet mező üres, külön csoportba sorolja.

Egészítsük ki a SELECT mezőlistáját egy magyarázó Karaktorsorozattal felhasználva az ACCESS If függvényét. Az If függvénynek három paraméter van. Az első paraméter egy logikai kifejezés, amelyet minden sorra kiértékel az SQL. Ha a logikai kifejezés igaz, az SQL a második paramétert hajtja végre, ha hamis, akkor a harmadik paramétert.

```
SELECT If(készlet < 15, 'kevés', 'elég'), Count(*) FROM anyag  
GROUP BY készlet < 15;
```

A megoldás nem jó. Az eredményből látható, hogy az üres mezőkre is azt kapjuk, hogy az If logikai kifejezése hamis, és az „elég” karaktersorozatot írja ki.

Fordítsuk meg az If függvényben a relációjelet. Természetesen akkor a GROUP BY záradékban is meg kell fordítanunk a relációjelet. Rendezzük le az eredménytáblát úgy, hogy az első sorban legyenek az üres mezőnek megfelelő „kevés” sorok, a másodikban a 15 egységnél kevesebb készlettel rendelkező „kevés” sorok, a harmadikban az „elég” sorok. Vegyük figyelembe, hogy logikai kifejezés szerinti rendezésnél a sorrend: üres, igaz, hamis.

```
SELECT If(készlet >= 15, 'elég', 'kevés'), Count(*)  
FROM anyag  
GROUP BY készlet >=15  
ORDER BY If(készlet >= 15, 'elég', 'kevés') DESC, készlet >= 15;
```

A rendezés előreveszi a két „kevés” sort az első szempont szerinti rendezés miatt, majd a második szempont szerint eldönti, hogy a két „kevés” sorból az üres mezők összesítéséből származó „kevés” kerüljön előre.

A rendezést világosan láthatjuk, ha kiegészítjük a lekérdezést a csoportosító feltétellel.

```
SELECT If(készlet >= 15, 'elég', 'kevés'), Count(*),  
       készlet >= 15  
FROM anyag  
GROUP BY készlet >=15  
ORDER BY If(készlet >= 15, 'elég', 'kevés') DESC, készlet >= 15;
```

Ha az eredménytáblában, a szöveges mezőben is meg szeretnénk különböztetni az üres készlet mezők darabszámát is, akkor a csoportosítás feltételébe is vegyük be a három csoportot. Az ORDER BY záradékot is elhagyhatjuk, ha kívánt rendezési szempont azonos a csoportosító mező növekvő sorrendjével.

```

SELECT Right(If(készlet Is Null, '1 nincs',
                If(készlet < 15, '2kevés', '3elég ')), 5),
        Count(*)
FROM anyag
GROUP BY If(készlet Is Null, '1 nincs',
            If(készlet < 15, '2kevés', '3elég '));

```

Az Is Null szabványos SQL kifejezés, arra a mezőre igaz, amelyik mező tartalma üres érték.

A GROUP BY záradékos eredménytábla rendezettségét a csoportosító mező értéke határozza meg (jelen esetben: „1 nincs”, „2kevés”, „3elég”). Mivel az első karakterre csak a rendezettség miatt volt szükségünk, és mindegyik hozzárendelt érték hathosszúságú (az „3elég” karaktersorozatot kiegészítettük egy betűközzel), ezért a megjelenítésnél a csoportosító mező értékéből csak jobbról az első öt karaktert írtuk ki az ACCESS Right függvényével.

Megjegyzések:

1. Ne bonyolítsuk felesleges HAVING záradékkal a lekérdezéseket.

Ha azt szeretnénk meghatározni, hogy hány anyagból van 15 egységnél kisebb készlet:

```

SELECT 'kevés', Count(*)
FROM anyag
GROUP BY készlet <15
HAVING készlet <15;

```

Figyelem, az üres készlet mező is kimarad az összegezésből!

Nem szerencsés megoldás, bár az ACCESS SQL szintakszis megengedi, hogy ebben az esetben a GROUP BY záradékot elhagyjuk.

```

SELECT 'kevés', Count(*)
FROM anyag HAVING készlet <15;

```

Van tehát aggregáló függvény a SELECT mezőlistájában, nincs GROUP BY záradék, de van HAVING záradék. Nem szerencsés megoldás, használatát nem javasoljuk, mert a fenti lekérdezés azonos azzal, mintha a HAVING záradék helyett WHERE záradékot használtunk volna.

```

SELECT 'kevés', COUNT(*)
FROM anyag
WHERE készlet <15;

```

2. A következő három SELECT utasítás szintaktikailag hibás.

```
SELECT 'kevés', készlet
FROM anyag HAVING készlet <15;
```

Nem csoportot ad vissza a lekérdezés, mert a SELECT mezőlistájában nincs aggregáló függvény, ezért HAVING záradék sem szerepelhet.

```
SELECT 'kevés', készlet, COUNT(*)
FROM anyag HAVING készlet <15;
```

Van aggregáló függvény, nincs GROUP BY záradék, van HAVING záradék, a lekérdezés egyetlen csoportot eredményez, a készlet nem jeleníthető meg.

```
SELECT If(készlet >= 15, 'elég', 'kevés'), készlet, COUNT(*) FROM
anyag GROUP BY If(készlet >= 15, 'elég', 'kevés');
```

Az eredmény tábla kettő vagy három sorból fog állni, aszerint, hogy van-e üres készlet mező, a készlet nem jeleníthető meg.

3. Ha szeretnénk minden anyagot megjeleníteni, amelyből kevés a készlet, nem kell sem GROUP BY záradék, sem HAVING záradék, hanem csak WHERE záradék:

```
SELECT anyag.*, 'kevés'
FROM anyag WHERE készlet < 15 OR készlet Is Null;
```

A NÖT készlet >= 15 kifejezés nem adja vissza az üres értékű készleteket!

Általános szabályként kimondhatjuk:

- A WHERE záradék a FROM záradékban megadott tábla sorait szűkíti.
- Ha a SELECT utasításban a GROUP BY záradék csoportosító ismervével (mező vagy kifejezés) csoportokat képezünk, akkor a SELECT mezőlistájában aggregáló függvény, konstans, a csoportosító mező vagy kifejezés, valamint ezekből képzett kifejezés szerepelhet csak. Ebből következik, ha a SELECT mezőlistájában szerepel egy mezőnév, akkor annak szerepelnie kell a GROUP BY mezőlistájában is. Fordítva az állítás nem igaz.
- A HAVING záradék a GROUP BY záradék csoportosító ismervével válogat a GROUP BY záradékkal kialakított csoportokból.

Megjegyzések:

1. Az ORDER BY záradékában az oszlopokat meghatározó kifejezések helyett a szabványos SQL megengedi, hogy az eredménytábla oszlopainak a sorszáma hivatkozzunk.

```
SELECT If(készlet >= 15, 'elég', 'kevés'), Count(*), készlet >= 15
FROM anyag
GROUPBY készlet >=15
ORDER BY If(készlet >= 15, 'elég', 'kevés') DESC, készlet >= 15;
```

A fenti lekérdezést tehát írhatjuk a következő formában is:

```
SELECT If(készlet >= 15, 'elég', 'kevés'), Count(*),
      készlet >= 15
FROM anyag
GROUPBY készlet >=15 ORDER BY 1 DESC, 2;
```

Az ACCESS környezetben az ilyen hivatkozás nem mindig ad helyes eredményt, ezért a könyvben a továbbiakban az oszlopokat meghatározó kifejezéseket fogjuk használni az ORDER BY záradékban.

2. Nem szabványos lehetőség, de az ACCESS SQL megengedi, hogy olyan mezőkre is rendezzünk, amelyek nem szerepelnek a SELECT utasítás mezőlistájában. Az ACCESS előállít egy közbenső eredménytáblát, aminek az oszlopait a teljes SQL utasítás elemzése alapján határozza meg, és így ennek lehet több oszlopa, mint a végső eredménytáblának. Ezt a közbenső eredménytáblát rendezi, majd ebből a SELECT utasításban lévő mezőlista szerint elvégzi a megjelenítést.

Gyakorló feladatok

1. Melyek a 2001. évben elkészített termékek kódja?
2. Határozzuk meg, hogy hány olyan megrendelés volt, amelyikben csak egyféle, amelyikben csak kétféle... stb. terméket rendeltek meg!
3. Keressük ki azoknak a termékeknek a kódját, amelyek legalább háromféle anyagból épülnek fel!

Megjegyzések:

1. Az ORDER BY záradékában az oszlopokat meghatározó kifejezések helyett a szabványos SQL megengedi, hogy az eredménytábla oszlopainak a sorszáma hivatkozzunk.

```
SELECT If(készlet >= 15, 'elég', 'kevés'), Count(*), készlet >= 15
FROM anyag
GROUPBY készlet >=15
ORDER BY If(készlet >= 15, 'elég', 'kevés') DESC, készlet >= 15;
```

A fenti lekérdezést tehát írhatjuk a következő formában is:

```
SELECT If(készlet >= 15, 'elég', 'kevés'), Count(*),
      készlet >= 15
FROM anyag
GROUPBY készlet >=15
ORDER BY 1 DESC, 2;
```

Az ACCESS környezetben az ilyen hivatkozás nem mindig ad helyes eredményt, ezért a könyvben a továbbiakban az oszlopokat meghatározó kifejezéseket fogjuk használni az ORDER BY záradékban.

2. Nem szabványos lehetőség, de az ACCESS SQL megengedi, hogy olyan mezőkre is rendezzünk, amelyek nem szerepelnek a SELECT utasítás mezőlistájában. Az ACCESS előállít egy közbenső eredménytáblát, aminek az oszlopait a teljes SQL utasítás elemzése alapján határozza meg, és így ennek lehet több oszlopa, mint a végső eredménytáblának. Ezt a közbenső eredménytáblát rendezi, majd ebből a SELECT utasításban lévő mezőlista szerint elvégzi a megjelenítést.

Gyakorló feladatok

1. Melyek a 2001. évben elkészített termékek kódja?
2. Határozzuk meg, hogy hány olyan megrendelés volt, amelyikben csak egyféle, amelyikben csak kétféle... stb. terméket rendeltek meg!
3. Keressük ki azoknak a termékeknek a kódját, amelyek legalább háromféle anyagból épülnek fel!

6. Paraméteres lekérdezés

1. Keressük meg, hogy „A” betűvel kezdődő nevű termék van-e az adatbázisban!

Az „A” betűvel kezdődő termékek neveit listázzuk ki!

```
SELECT név  
FROM termék  
WHERE név LIKE 'A*';
```

Ugyanezt az eredményt kapjuk meg a Left(név, 1) függvény használatával is, amely balról az első karakterét adja vissza a név mezőnek.

```
SELECT név  
FROM termék  
WHERE Left(név, 1)='A';
```

A kettő között csak annyi a különbség, hogy a LIKE művelet szabványos SQL művelet, míg a Left függvény az ACCESS függvénye, bár minden SQL megvalósításban elérhető hasonló függvény a befogadó környezetből.

2. Keressük meg, hogy előre nem definiált, a lekérdezés idején megadott karaktersorozattal kezdődő nevű termék van-e az adatbázisban!

Amennyiben a következő lekérdezésben a „B” betűvel kezdődő termékek neveit szeretnénk kilistázni, át kellene írni a SELECT utasítást. Ha el akarjuk kerülni a SELECT utasítás állandó módosítását, paramétereket használhatunk a lekérdezésben. A paraméter egy olyan név, amihez nem tartozik mező a FROM záradékban megadott táblákban, így az utasítás kiértékelésénél nincs az adatbázisból hozzárendelhető érték, a kiértékelés előtt egy párbeszéd ablakból fogja kérni az ACCESS SQL a kiértékelendő a paraméter aktuális, csak az adott lekérdezésre érvényes értékét. (A kis és nagybetű között az ACCESS nem tesz különbséget.)

```
SELECT név  
FROM termék  
WHERE név LIKE [keresett érték] &
```

Mivel a paraméter neve (keresett érték) két szóból álló egyetlen név, azért, hogy ezt egy szintaktikai egységként kezelje az SQL, szögletes zárójelbe kell tenni a nevet.

Vegyük észre, hogy a paraméter (keresett érték) tetszőleges hosszú lehet. Üres karaktersorozatot megadva, a teljes listát adja vissza. A lekérdezés a paraméter értékével kezdődő neveket adja vissza.

Ha csak arra vagyunk kíváncsiak, hogy az adott paraméterrel kezdődő nevű termék létezik-e az adatbázisban, és ha létezik, hány van belőle:

```
SELECT [keresett érték] AS [mit keresek], COUNT(*) AS
darabszám, If{darabszám = 0, 'nincs', 'van'} AS [van-e] FROM
termék
WHERE név LIKE [keresett érték] &
```

A mező nevet követő AS kulcsszó utáni név a lekérdezés eredménytáblájában az oszlop neveként fog megjelenni.

Az ACCESS tervező rácsán a lekérdezést így fogalmazhatjuk:



<

Mező:				
Tábla:		termék		termék
Összesítés:	Expression	Count	Expression	Where
Rendezés:				
"egjefenítés:	0	0	0	
Feltétel:				Like [keresett érték] &
vagy:				

Ha a megadott feltételnek a TERMÉK táblában egyetlen sor sem felel meg, az eredménytáblának akkor is lesz egy sora, ahol a darabszám egyenlő nullával.

Gyakorló feladatok

1. Készítsünk éj olyan paraméteres lekérdezést, amelyik a paraméterrel megadott partnerkódra megjeleníti a megrendelőnek a 2001. évben történt összes rendeiésszámát és a megrendelés dátumát!
2. Készítsünk egy olyan paraméteres lekérdezést, ahol paraméterekkel lehet megadni azt az időintervallumot, amely meghatározza a rendelés dátumát! Írassuk ki a partner kódját, a rendelésszámot és a megrendelés dátumát! Az eredménytábla a rendelési dátum csökkenő, azon belül partnerkód növekvő sorrendjében jelenjen meg!
3. Készítsünk egy lekérdezést, amelyik paraméterrel megadott termékkódra kiírja az adott termékből az elmúlt hónapban elkészült összes darabszámot!

7. Táblák összekapcsolása, direktszorzat

1. Határozzuk meg, hogy egy adott termék milyen anyagokból épül fel! A termékeknek a nevét és a bennük lévő anyagoknak az azonosítóját írassuk ki!

A termék neve a TERMÉK táblában van, az anyag azonosítója pedig mind a SZERKEZET, mind az ANYAG táblában benne van. Mivel az anyag nevére, a feladat megfogalmazása szerint, nincs szükségünk, a lekérdezésben elég a TERMÉK és a SZERKEZET táblát használna.

```
SELECT termék.*, szerkezet.*  
FROM termék, szerkezet;
```

A lekérdezés FROM záradéka határozza meg, hogy milyen táblák fognak szerepelni a lekérdezésben, és az eredmény táblának annyi sora lesz, mint amennyi a TERMÉK tábla sorainak a száma szorozva a SZERKEZET tábla sorainak a számával. A FROM záradék a direktszorzat definíciója szerint minden sort minden sorral párosít.

Azért, hogy világosan lássuk az eredmény szerkezetét, mindkét táblából csak a termék kódját írassuk ki.

```
SELECT termék.kód, szerkezet.kód  
FROM termék, szerkezet;
```

Minden TERMÉK táblából származó kód mellett meg fog jelenni minden kód a SZERKEZET táblából. A lekérdezés előállítja a FROM záradékban megadott táblák direktszorzatát, és ebből kiírja a két kód mezőt. Az eredmény nagy része teljesen értelmetlen párokat ad vissza.

Ha „értelmessé” akarjuk tenni az eredménytáblát, a drrektszorzatból csak a kulcs - külső kulcs kapcsolat által meghatározott párokat hagyjuk meg.

A TERMÉK táblában a kód mező a kulcs, ami azt jelenti, hogy egy megadott kód a TERMÉK táblából egyetlen sort, egy értékármast határoz meg (kód, név, ár). A SZERKEZET tábla kulcsa a kód mezőből és azonosító mezőből képzett értékpár, ami azt jelenti, hogy ahhoz, hogy egyetlen sort tudjunk kiválasztani a táblából egy kódot és egy azonosítót kell megadni. A SZERKEZET táblában a kód mező külső kulcs, mert a SZERKEZET tábla kód mezőjének és a TERMÉK tábla kód mezőjének közös az értékhalmaza, és a TERMÉK táblában a kód mező kulcs. Annak, hogy a kód mező a SZERKEZET tábla kulcsának a része, nincs jelentősége. A SZERKEZET táblában egy kód értékhez általában nem egyetlen sor tartozik. Egy kód értékkel azokat a sorokat tudjuk kiválasztani, amelyek egy termékben lévő anyagokat határozzák meg (kivételesen, ha a termék egyféle

anyagból áll, akkor választ ki csak egy sort, de általában a kiválasztás több sort eredményez). Röviden ezt úgy mondjuk, a TERMÉK és SZERKEZET tábla között 1 : N (egy a több) kapcsolat van, és ezt a kapcsolatot a kód mezők értékei hordozzák.

Mindebből az következik, hogy a két táblából „értelmes” összekapcsolt táblát úgy kapunk, ha a SZERKEZET tábla soraihoz olyan sorokat kapcsolunk a TERMÉK táblából, ahol a két táblából származó kód értéke megegyezik. Az így ösz- szekapcsolt tábla egy sora egy termékről adja meg a kód, név, ár mező értékét és a terméket felépítő valamelyik anyagának az azonosítóját és mennyiségét.

```
SELECT termék.kód, szerkezet.kód
FROM termék, szerkezet
WHERE termék.kód = szerkezet.kód;
```

Ennek a táblának annyi sora lesz, mint ahány sora van a SZERKEZET táblának van, ugyanis a SZERKEZET táblának minden sora egy TERMÉK táblabeli sónál tud párt alkotni. Az egy sorban lévő két kód mező értéke azonos.

Írjuk be a SELECT mezőlistájába a TERMÉK táblából származó nevet és a SZERKEZET táblából származó azonosítót.

```
SELECT név, termék.kód, szerkezet.kód, azonosító
FROM termék, szerkezet
WHERE termék.kód = szerkezet.kód;
```

Hagyjuk ki a két középső oszlopot, azaz a két kód mezőt, és megkapjuk a feladat megoldását. Minden termék neve mellett kiírásra kerül a benne lévő anyag . amelyik azonosítója. Az eredmény táblának annyi sora lesz, ahány sora van a SZERKEZET táblának. (Feltéve, hogy minden olyan terméknek, aminek van szerkezeti leírása, megvan a TERMÉK táblában is a megfelelő sora, azaz jó az adatbázis, más szóval az adatbázis felépítésénél vigyáztunk az integritási szabá- vokra.)

```
SELECT név, azonosító
FROM termék, szerkezet
WHERE termék.kód = szerkezet.kód;
```

Érdemes megfigyelni, hogy az összekapcsolt táblánál már „elrontottuk a harmadik normálformát”. Az eredménytáblának a kulcsa ugyanaz, mint a SZERKEZET táblának, a SZERKEZET táblából származó kód és azonosító mező, de a 'Zerkezet.kód meghatározza a termék.kód értéket, és a termék.kód érték meghatározza a termék.név és a termék.ár értékét, az eredménytábla tehát redundáns. Ugyanakkor az eredménytábla sokkal könnyebben átlátható, mint az eredeti két tábla (lehet persze még sokat javítani az eredménytábla használhatóságán!).

Az összekapcsolt táblában egy terméknek a TERMÉK táblából származó kódja és az általa meghatározott név és ár annyiszor fog szerepelni, ahányféle anyagból áll a termék.

TERMÉK			SZERKEZET		
név	ár	kód	kód	...	azonosító
ni	ál	ki	ki		all
ni	ál	ki	ki		al2
ni	ál	ki	ki		aln
n2	á2	k2	k2		a21
n2	á2	k2	k2		a22
n2	á2	k2	k2		a2m

2. A termék nevét és a benne lévő anyagok nevét írjuk ki!

A megoldás lépései:

1. Mit kell kiírni, és ezek melyik táblákban találhatóak?
név mező a TERMÉK táblából és a neve mező az ANYAG táblából
2. Milyen kapcsolat van a táblák között?
jelenleg semmilyen (a termékek és az anyagok között N : M logikai viszony van)
3. Hogyan lehet kapcsolatba hozni a szükséges táblákat, melyik táblán (táblákon) keresztül lehet őket „kapcsolatba hozni”, azaz hogyan lehet olyan táblasorozatot felépíteni, ahol az egymást követő táblák 1 : N, illetve N : 1 kapcsolatban vannak egymással?

TERMÉK : SZERKEZET 1 : N, SZERKEZET : ANYAG N : 1

A SZERKEZET táblában egy termékkód a TERMÉK tábla termékkódján keresztül meghatároz egy terméknevet, és egy anyagazonosító a SZERKEZET táblából az ANYAG tábla anyagazonosítóján keresztül meghatároz egy anyagnevet.

A feladatot megoldó lekérdezésben szükséges tehát a SZERKEZET tábla is. A három tábla „értelmes” összekapcsolásából határozhatjuk meg az összetartozó termék- és anyagneveket. Bár a SZERKEZET táblából nem kell kiíratnunk semmit, de a TERMÉK és az ANYAG táblában lévő összetartozó mezőértékek összekapcsolásához szükségünk van a SZERKEZET táblára.

```
SELECT név, termék.kód, szerkezet.kód,  
       szerkezet.azonosító, anyag.azonosító, neve  
FROM termék, szerkezet, anyag  
WHERE termék.kód = szerkezet.kód  
       AND szerkezet.azonosító = anyag.azonosító;
```

A SELECT mezőlistájából a felesleges mezők elhagyásával:

```
SELECT név, neve  
FROM termék, szerkezet, anyag WHERE termék.kód =  
       szerkezet.kód  
       AND szerkezet.azonosító = anyag.azonosító;
```

Ha konkrétan az „AA1” kódú termékben lévő anyagok nevét akarjuk kiírni:

```
SELECT név, neve  
FROM termék, szerkezet, anyag  
WHERE termék.kód = szerkezet.kód  
       AND szerkezet.azonosító = anyag.azonosító  
       AND termék.kód = 'AAT';
```

Ha arra a kérdésre keressük a választ, hogy a „112” azonosítójú anyag milyen :
ermékekbe épül be, és a termék nevét és az anyag nevét szeretnénk kiírni, „gvanúgy a
TERMÉK, SZERKEZET és az ANYAG táblára van szükségünk. Az „értelmessé” tevő
összekapcsolás feltétele is ugyanaz. Egyetlen különbség, hogy nem a termék.kód mezőre,
hanem az anyag.azonosító mezőre kell a további szűrő feltételt megadni.

```
SELECT neve, név  
FROM termék, szerkezet, anyag  
WHERE termék.kód = szerkezet.kód  
       AND szerkezet.azonosító = anyag.azonosító AND  
       anyag.azonosító = 112;
```

A két utóbbi SQL utasítás szerkezete azonos. A mezőnevek sorrendjében és a WHERE
záradék harmadik relációjában van csak különbség. A mezőnevek sorrendjét csak azért
váltottuk meg, hogy a lekérdezés eredményét megtekintő számára érthetőbb legyen az
eredmény információtartalma. (Ez nagyon fontos kérdés, de ez már szervezési és nem SQL
kérdés!)

Bár az SQL szintaxis szerint a WHERE záradékban szereplő három reláció között
látszólag nincs semmi különbség, érdemes mindig megtenni a következő szétválasztást:

7. Táblák összekapcsolása, direktszorzat

„Értelmessé tevő” feltételek:

termék kód = szerkezet.kód

AND szerkezet.azonosító = anyag.azonosító

Ezek a feltételek arra szolgálnak, hogy a különböző táblákból az összetartozó sorokat összekapcsolják, és az összekapcsolás által egy új, tartalmát tekintve helyes tábla álljon elő (szokásos elnevezése: egyenlőségen alapuló természetes ősz- szekapcsolás).

„Szűrő” feltételek:

termék.kód = "AA1"

anyag.azonosító = 112

Az összekapcsolás által kialakult egyetlen tábla soraiból kiválogatja a feladatnak megfelelő sorokat, szűri vagy más szóval függőleges irányban szűkíti az ősz- szekapcsolt táblát.

Az „értelmes” összekapcsolás a termék.kód = szerkezet.kód AND szerkezet.azonosító = anyag.azonosító feltétel alapján:

TERMÉK

SZERKEZET

ANYAG

név		kód	kód	...	azonosító	azonosító		neve
ni		ki	ki		all	all		nel 1
ni		ki	ki		al2	al2		ne 12
ni		ki	kin		aln	aln		neln
n2		k2	k2		a21	a21		ne21
n2		k2	k2		a22	a22		ne22
n2		k2	k2		a2m	a2m		ne2m

ni ◀----- ki <

(ki; all)

> all

> nel 1

A táblázat az összetartozó termékneveket és azonosítóikat szemlélteti:

név	neve
ni	nel 1
ni	nel2
ni	neln
n2	ne21
n2	ne22
n2	ne2m

Ha al terméknevek nem egyediek, - és mivel a név nem kulcs a TERMÉK táblá- ~ ■ ha megszorító feltételt nem teszünk a TERMÉK tábla név mezőjére, akkor re— biztos, hogy egyediek - ez az eredménytábla félrevezető lehet. Célszerű ezért, a terméknevekre és anyagnevekre nem biztosítottuk, hogy egyediek legyenek, a e—ek kódokat és az anyagazonosítókat is minden esetben megjeleníteni!

Az összekapcsolás után a termék.kód = „AA1” feltétel alapján „szűrt” tábla:

TERMÉK		SZERKEZET		ANYAG			
név		kód	kód		azonosító	azonosító	neve
		AA1	AA1		al	al	nel
		AA1	AA1		a2	a2	ne2
		AA1	AA1		an	an	nen

Az „AA1” termékkódhoz tartozó terméknév és anyagazonosítók tehát:

név	neve
n	nel
n	ne2
n	nen

Ez az eredmény, - függetlenül attól, hogy a terméknév egyedi vagy sem - min- helyes eredményt ad, csak az „AA1” termékkódú termék felépítéséhez szükig es anyagok lesznek benne felsorolva.

gyakorló feladatok

- \ Melyek azok a megrendelt termékek, amelyek még nincsenek készen, és a ter- —ek előállításához kell az „alátét” nevű anyag? írássuk ki a termék nevét nő- ekvő sorrendben!
- l írássuk ki rendelésszámonként a 2001. januárjában megrendelt összes termék eevét és darabszámát!
- | tMÉyek azok a megrendelések, amelyeknek az összértéke nagyobb, mint _ 00 Ft? írássuk ki a rendelés számát, a rendelés dátumát és összértékét!

8. Beágyazott SELECT utasítás

1. Keressük meg azoknak a termékeknek a nevét, amelyeknek az előállításához kell „113”-as azonosítójú anyag!

Először csak a „113”-as azonosítójú anyagot tartalmazó termékek kódját keressük meg. A termék kódja és a hozzá szükséges anyagok azonosítója benne van a SZERKEZET táblában.

```
SELECT kód
FROM szerkezet
WHERE azonosító = 113;
```

KÓD_113{kód}

Egy kódérték csak egyszer fog szerepelni az eredménylistában, mert egy adott kódérték legfeljebb egyszer fordulhat elő egy adott azonosítóval, mivel a két mező együttesen a tábla kulcsa.

Ha a termék kódja helyett a termék nevét szeretnénk kiírni, mivel a név a TERMEK táblában van, szükséges a TERMEK tábla is.

```
SELECT név
FROM termék, szerkezet WHERE termék.kód = szerkezet.kód
AND azonosító = 113;
```

Ha az adatbázisban vannak azonos nevű termékek, amelyekben van „113”-as azonosítójú anyag, akkor a közös nevük többször is szerepelhet. Elvileg ez lehetséges, mivel a kód és nem a név a TERMÉK tábla kulcsa.

TERMÉK

SZERKEZET

név	kód	kód	azonosító
jó név	x----- b.	= -4— x	113-4 -----
nem jó	P —	— p	q -4 -----

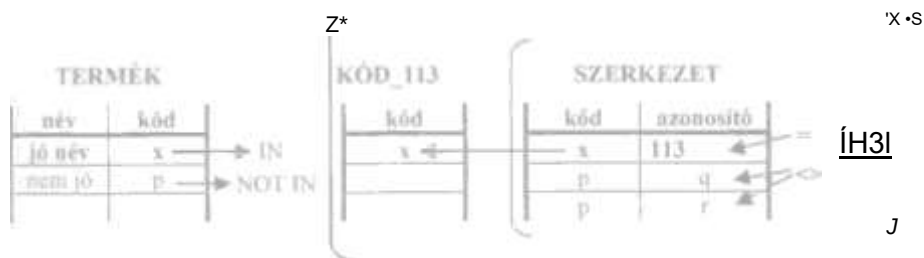
A SZERKEZET táblából semmit sem akarunk kiírni, csak azért volt rá szükségünk, hogy eldönthessük, hogy van-e a termékben 113 azonosítójú anyag. Fogalmazhatjuk ezért úgy is a feladatot, hogy a TERMÉK táblából azokat a neveket írassuk ki, amelyeknek a kódja benne van a KÓD 113 táblában lévő kódok között.

```
SELECT név
FROM termék
WHERE kód IN (SELECT kód
FROM kód_113);
```

A belső SELECT utasítás helyett írható maga az első utasítás is.

```
SELECT név
FROM termék
WHERE kód IN (SELECT kód
               FROM szerkezet
               WHERE azonosító = 113);
```

Mindkét esetben a belső SELECT utasítás feladata egy termékkódokból álló halmaz meghatározása. A halmaz elemeinek a meghatározása nem függ semmi- lve a külső értéktől, minden információ, ami a halmaz elemeinek az összegyűjtéséhez kell, rendelkezésre áll a belső SELECT utasításban. Az SQL ezért megteheti, és meg is teszi, hogy a külső SELECT utasítás kiértékelése előtt a belső SELECT utasítást kiértékelje. A belső SELECT utasítás így csak egyszer, a külső SELECT utasítás előtt fog kiértékelődni.



2. A „113” azonosítójú anyagokat tartalmazó termékeknek nemcsak a nevét, hanem a teljes szerkezetét, azaz minden anyagának az azonosítóját is írassuk ki!

Az előző feladathoz képest az az eltérés, hogy a **TERMÉK** és a **SZERKEZET** táblák „értelmes” összekapcsolásával készített tábla sorainak csoportjait kell kiíratni, azonban egy sorról csak közvetetten lehet meghatározni, hogy hozzátartozik-e a kiíratandóakhoz, önmagában a sorról ez nem dönthető el. Egy sort akkor kell kiíratni, ha az adott sor olyan, hogy létezik egy olyan sor a **SZERKEZET** táblában, hogy a benne lévő termék kód azonos a vizsgált sorban lévő termékkóddal, és az anyag azonosítója egyenlő 113-mal.

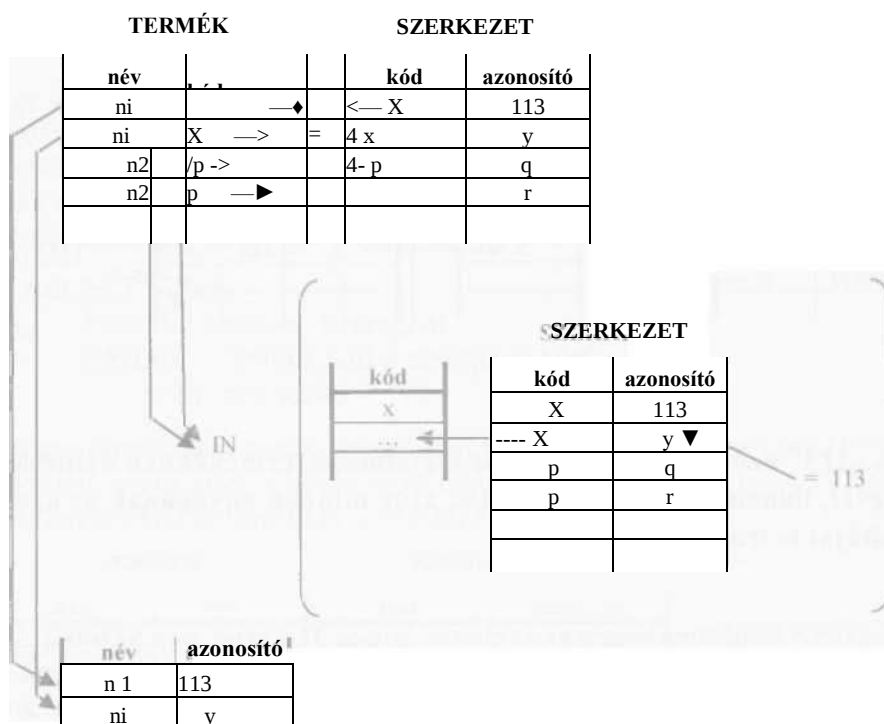
A megoldáshoz először válasszuk ki az összes olyan terméknek a kódját, amelyekben van „113”-as kódú anyag. Ehhez elég a **SZERKEZET** táblát felhasználni. Majd a **TERMÉK** és a **SZERKEZET** táblából készített értelmesen összekapcsolt tábla minden sorról eldöntjük, hogy a kigyűjtött kódot tartalmazza-e.

```

SELECT név, azonosító
FROM termék, szerkezet
WHERE termék.kód = szerkezet.kód
      AND termék.kód IN (SELECT kód
                        FROM szerkezet
                        WHERE azonosító = 113);

```

A belső SELECT utasítás csak egyszer hajtódik végre, mivel a kiértékeléséhez nem szükséges semmilyen olyan információ, ami a külső SELECT utasításból származik. A belső SELECT utasításban a mezőneveket nem kell minősíteni, azok alapértelmezés szerint a belső SZERKEZET táblához tartoznak.



Más, bonyolultabb gondolatmenettel is eljuthatunk ugyanehhez a megoldáshoz. A két táblából készített értelmes összekapcsolás után egy adott sort akkor írathatjuk ki, ha a sor olyan termékhez tartozik, amelyiknek a szerkezetében van „113”-as azonosítójú anyag is. Másképpen fogalmazva: egy adott termékhez gyűjtsük ki SZERKEZET táblából az összes sorát, és vizsgáljuk meg, hogy találunk-e benne „113”-as azonosítót.

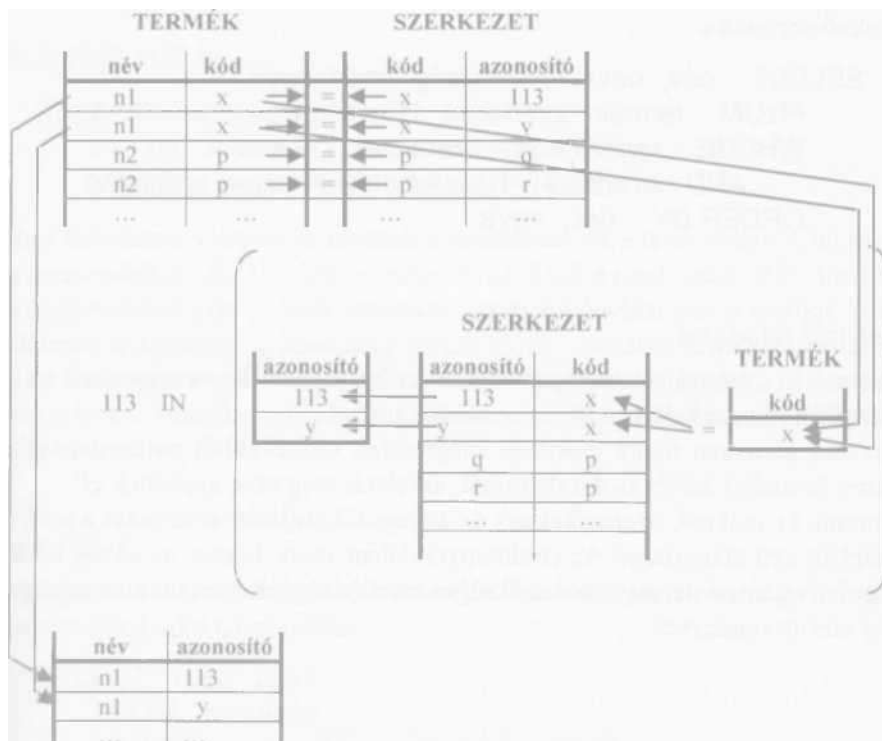
```

SELECT név, azonosító
FROM termék, szerkezet
WHERE termék.kód = szerkezet.kód
      AND 113 IN (SELECT azonosító FROM szerkezet
                  WHERE kód = termék.kód);

```

A belső SELECT utasítás önmagában nem értékelhető ki a lekérdezés elején, mert a termékkód értékét a külső SELECT utasítás határozza meg. A külső SELECT utasításban a TERMÉK és SZERKEZET táblának értelmesen összekapcsolt minden sorának a kiértékelésénél ki kell értékelni a belső SELECT utasítást. Természetesen minden SQL megvalósítás igyekszik az ilyen, nagy erőforrás-igényű parancs végrehajtását optimalizálni, pl. belső rendezéssel, indexeléssel stb., de ez a megoldás mindenképpen nagyon drága lesz. Ha nem szükséges, kerüljük az ilyen belső SELECT utasításokat.

Azt, hogy a példatárban szereplő megoldásváltozatból melyik az optimális, csak az adott környezet ismeretében lehet eldönteni.



3. A „113” azonosítót tartalmazó termékek teljes szerkezetének a kiírásánál a termék nevét, a benne lévő anyag nevét, a belőle szükséges mennyiséget és a mértékegységet is írassuk ki!

A feladat megoldásához az ANYAG táblára is szükségünk lesz, tekintve, hogy az anyag neve és mértékegysége ebben a táblában található.

A legnagyobb tábla a SZERKEZET tábla. Valószínűleg gyorsabb lesz a feldolgozás, ha először a SZERKEZET tábla sorait csökkentjük, csak azokat hagyjuk meg, amelyek a feladatban meghatározott termékek szerkezetét írják le.

```
SELECT kód, azonosító, mennyiség
FROM szerkezet
WHERE kód IN (SELECT kód
              FROM szerkezet
              WHERE azonosító = 113);
-> SZERKEZET_113{kód, azonosító, mennyiség}
```

Utána ezt a táblát felhasználva a TERMÉK, a SZERKEZET113 és az ANYAG tábla értelmes direktzorzatának kiírása adja a feladat megoldását. Rendezzük az eredmény táblát a termék neve és azon belül az anyag neve szerint növekvő sorrendbe.

```
SELECT név, neve, mennyiség, mért_egys
FROM termék, szerkezet_113, anyag WHERE termék.kód =
szerkezet_113.kód
AND szerkezet_113.azonosító = anyag.azonosító ORDER
BY név, neve;
```

Gyakorló feladatok

1. írassuk ki azoknak a termékeknek a nevét és összetételét, amelyeknek az előállításához nem kell a „113” azonosítójú anyag!
2. írassuk ki a mai napra esedékes megrendelt termékekből partnerkódonként azon termékek nevét és darabszámát, amelyek még nem készültek el!
3. írassuk ki azoknak a termékeknek az anyagszükségletét, amelyeket a jövő hónapban kell elkészíteni! Az eredménytáblában az év, hónap, az anyag neve és kódja, valamint az anyagból szükséges mennyiség jelenjen meg az anyag neve szerint rendezve!

9. Csoportok kiválogatása

Határozzuk meg, hogy melyik megrendelésekben szerepel a „BC1” és a „BC9” kódú termék is! (elég a megrendelés számát kiíratni)!

(Mindkét tennék szerepeljen ugyanabban a megrendelésben, azaz a rend szám- uk azonos legyen!)

Ami nagyon nem jó:

```
SELECT rend_szá  
      FROM rendelés  
      WHERE kód ='BC1' AND kód ='BC9';
```

Ez a lekérdezés üres halmazt ad vissza, mivel a lekérdezés soronként értékeli, hogy az adott sor kielégíti-e a megadott feltételt. Egyetlen sor sem tehet eleget annak a feltételnek, hogy egyszerre legyen igaz az is, hogy a kód „BC1”-gyel egyenlő és ugyanakkor „BC9”-vel is egyenlő.

Ami szintén nem jó:

```
SELECT rend_szá  
      FROM rendelés  
      WHERE kód = 'BC1' OR kód = 'BC9';
```

Ez a lekérdezés visszaadja azoknak a rendeléseknek a rend szám-át, amelyekben megrendeltek „BC1 ” kódú terméket és azoknak a rend szám-át is, amelyekben megrendeltek „BC9” kódú terméket. Az eredményben benne lesznek azok a rendelések is amelyek - a feladatnak megfelelően - mindkét terméket megrendelték, de általában ez az eredményhalmaz sokkal nagyobb, mint amit a feladat kért, ugyanis benne vannak azok a rend szám-ok is, amelyekben csak az egyiket rendelték meg.

L Megoldás

Válogassuk le a rendelés táblából azokat a sorokat, amelyek vagy az egyik, vagy a másik kódot tartalmazzák.

```
SELECT rend_szá  
      FROM rendelés  
      WHERE kód ='BC1' OR kód = 'BC9';
```

Az eredmény halmazban minden rend szám vagy egyszer, vagy kétszer szerepel aszerint, hogy csak a „BC1”-et vagy csak a „BC9”-et rendelte meg, illetve mindkettőt megrendelte. Csoportosítsuk rend szám szerint, és számoljuk össze, hogy egy csoportba hány sor került.

```
SELECT rend_szá, COUNT(*)
  FROM rendelés
 WHERE kód = 'BC1' OR kód = 'BC9'
 GROUP BY rend_szá;
```

Ezekből a sorokból a feladatnak csak azok a sorok felelnek meg, ahol a COUNT(*) = 2.

```
SELECT rend_szá, COUNT(*)
  FROM rendelés
 WHERE kód = 'BC1' OR kód = 'BC9'
 GROUP BY rend_szá
 HAVING COUNT(*) = 2;
```

Az megjelenítendőek listájából a COUNT(*) el is hagyható.

```
SELECT rend_szá
  FROM rendelés
 WHERE kód = 'BC1' OR kód = 'BC9'
 GROUP BY rend_szá
 HAVING COUNT(*) = 2;
```

II. Megoldás

Gyűjtsük ki egy táblázatba azokat a rend szám-okat, amelyekben „BC1”-et rendeltek meg, és egy másikba, amelyekben „BC9”-et rendeltek meg.

```
SELECT rend_szá
  FROM rendelés
 WHERE kód = 'BC1';
```

-> R_ELSŐ{rend_szá}

```
SELECT rend_szá
  FROM rendelés
 WHERE kód = 'BC9';
```

-> R_MÁSOD{rend_szá}

Azok a rend_szá-mok felelnek meg a feladatnak, amelyek mindkét táblában benne vannak.

Készítsünk egy olyan táblázatot, amely minden lehetséges párosítást tartalmaz a két táblázatból.

```
SELECT r_első.rend_száma, r_másod.rend_száma
FROM r_első, r_másod;
```

Az első rend száma az mutatja, hogy ebben megrendelték a „BC1”-et, a második pedig azt, hogy megrendelték a „BC9”-et. Azok a sorok lesznek jók, ahol a két rend száma megegyezik, azaz mindkettőt megrendelték ugyanabban a rendelésben.

```
SELECT r_első.rend_száma FROM r_első, r_másod WHERE
r_első.rend_száma = r_másod.rend_száma;
```

Vegyük észre, hogy a megoldást az első.rendszám és r_másod.rend_száma oszlopokban lévő rendszámok közös részeként, metszeteként kaptuk meg.

III. Megoldás

Hatékonyabb megoldás, ha végigmegyünk az R ELSŐ táblázat minden során, és csak azokat a rend száma-okat írjuk ki, amelyek benne vannak az RMASOD iablázatban. Ezzel a megoldással ugyanis elkerüljük a direktszorzat készítését, és annak a WHERE záradékban lévő feltétellel való szűrését.

```
SELECT rend_száma
FROM r_első
WHERE rend_száma IN (SELECT rend_száma
FROM r_másod);
```

Ha nem akarjuk az R ELSŐ és az R MÁSOD lekérdezést előre elkészíteni, -*natjuk*:

```
SELECT rend_száma
FROM rendelés
WHERE kód = 'BC1'
AND rend_száma IN (SELECT rend_száma
FROM rendelés WHERE kód =
'BC9');
```

A belső SELECT utasításban a kód elé nem kell minősítés. Ha a belső FROM záradékban lévő táblában megtalálható egy mező - függetlenül attól, hogy van _ en mező a külső SELECT utasításban is - a mezőt a belső táblához tartozónak ■ ?gja tekinteni az SQL. Az utasítás végrehajtása során a RENDELÉS táblát kétszer kell végigolvasni, először a belső SELECT utasítás végrehajtásánál, ami készít egy rendelésszámokból álló halmazt, majd a külső SELECT utasítás végrehajtásánál.

IV. Megoldás

AII. megoldásnál sem szükséges az előkészítő lekérdezések elvégzése.

```
SELECT első.rend_szá  
      FROM rendelés első, rendelés másod  
      WHERE első.kód ='BC1' AND másod.kód = 'BC9'  
            AND első.rend szám = másod.rend_szá
```

A RENDELÉS táblát kétszer használtuk fel ugyanabban a SELECT utasításban, ezért minden mezőt minősíteni kell, hogy azonosítani lehessen, hogy az adott mező melyik RENDELÉS táblához tartozik.

Ne felejtsük el! Minden olyan megoldás, amelyik egy táblát önmagához kapcsol hozzá, nagyobb táblázatok esetén meglehetősen erőforrás-igényes.

V. Megoldás

Térjünk vissza a III. megoldáshoz.

A belső SELECT utasítás azokat a rend szám-okat adja, amelyekben megrendeltek „BC9” kódú terméket. A feladat megfogalmazásában a „BC1” és a „BC9” teljesen egyenértékű. Ezért megoldhatjuk úgy is a feladatot, hogy végigmegyünk a RENDELÉSFEJ tábla minden során, és megnézzük, hogy ebben a megrendelésben megrendeltek-e „BC1” kódú és „BC9” kódú terméket is.

```
SELECT rend_szá  
      FROM rendelésfej  
      WHERE rend_szá IN (SELECT rend_szá  
                          FROM rendelés  
                          WHERE kód = 'BC1')  
            AND rend_szá IN (SELECT rend_szá  
                              FROM rendelés  
                              WHERE kód = 'BC9');
```

Vegyük észre, hogy ennél a megoldásnál a RENDELÉSFEJ táblát is felhasználtuk, és lényegében ugyanúgy elkészül a két rendelésszám halmaz, mégis, bízva a lekérdező rendszer optimalizálásában, ez a megoldás nagyon hatékony lehet.

VI. Megoldás

Az előző megoldások variálásából további változatokat készíthetünk. (Ezért szép az SQL!).

Az R ELSŐ lekérdezésben kigyűjtöttük azokat a rendelés számokat, amelyekben „BC1” kódú terméket is megrendeltek. Ennek a felhasználásával gyűjtjük ki

a RENDELÉS táblából az ezekhez a rendeléshez tartozó összes megrendelt termék kódját.

```
SELECT rendelés.rend_száma, kód
FROM rendelés, r_első
WHERE rendelés.rend_száma = r_első.rend_száma;
      -> R_E_TELJES{rend_száma, kód}
```

Az R E TELJES táblázat olyan megrendeléseknek a teljes terméklistáját tartalmazza, amelyekben biztosan benne van a „BC1” kódú termék is. Ebből kell kiválogatni az „BC9” kódú terméket is megrendelt megrendeléseket.

```
SELECT rend_száma
FROM r_e_teljes
WHERE kód = 'BC9';
```

A két lekérdezést összevonhatjuk.

```
SELECT rendelés.rend_száma
FROM rendelés, r_első
WHERE rendelés.rend_száma = r_első.rend_száma
      AND kód = 'BC9';
```

Vegyük észre, hogy itt nem kerestük ki az összes olyan megrendelést, amelyikben „BC9” kódú terméket is megrendeltek, viszont a második lépésben újra végigolvastuk a RENDELÉS táblát, hogy a „BC1” kódú terméket megrendeltek minden megrendelését végignézzük. Ha jobban belegondolunk, bár más gondolatmenettel, a III. megoldáshoz jutottunk.

I

VII. Megoldás

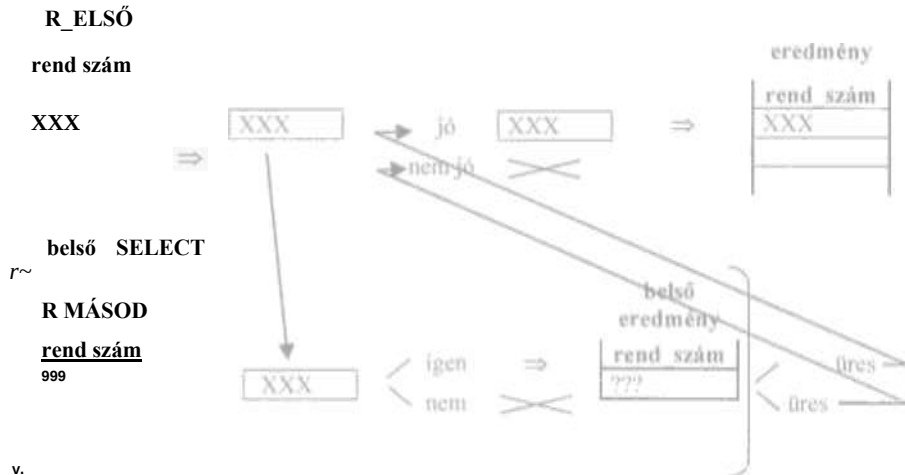
A III. megoldás első SELECT utasítását úgy is fogalmazhatjuk, hogy az R ELSŐ táblából a rend_száma-t akkor fogadjuk el, ha létezik vele egyenlő rend száma az R MÁSOD táblában. Használjuk az EXISTS predikátumot:

```
SELECT rend_száma
FROM r_első
WHERE EXISTS (SELECT rend_száma
              FROM r_másod
              WHERE rend_száma = r_első.rend_száma);
```

A belső SELECT utasításban a rend száma az R MÁSOD táblához tartozik, az r_első.rend_száma pedig a külső SELECT utasításban lévő R ELSŐ táblához. Ennek következtében a belső SELECT önmagában nem értékelhető ki, függ a külső SELECT utasításban vizsgált sor aktuális értékétől. A végrehajtás során a

belső SELECT utasítás a külső SELECT-ben kiválasztott minden sorra ismételt ki fog értékelődni.

külső SELECT



Megjegyzés

Mind a hét megoldás jó eredményt ad, de természetesen a hatékonyságuk különböző. A keresés hatékonysága függ az adott szoftvertől és attól, hogy hány és milyen nagy táblához kell hozzányúlni, kérünk-e direktszorzat készítest, milyen index táblák léteznek és milyen a megrendelések összetétele.

A lekérdezés készítésénél néhány szempontot azért érdemes figyelembe venni:

- a direktszorzat készítése mindig drága;
- a belső lekérdezés, ha az függ a külső lekérdezéstől, azaz nem készíthető el a külső lekérdezés eh égzése előtt, igen drága;
- a rendezés is időigényes feladat;
- feleslegesen ne használjunk EXISTS predikátumokat;
- indexekkel sok időt takaríthatunk meg;
- használjuk az adott SQL FROM záradékában a táblák összekapcsolására vonatkozó lehetőségeket (ezeket a példatárban nem tárgyaljuk, mert, bár az SQL szabvány foglalkozik velük, elég sok a különböző SQL megvalósítások között az eltérés);
- és mindenekelőtt, figyelmesen olvassuk el a gyártó cég optimalizálásra vonatkozó javaslatait!

Gyakorló feladatok

1. Melyik az a termék, amelynek az előállításához legfeljebb 10 egységnyi „112” azonosítójú és legfeljebb 5 egységnyi „113” azonosítójú anyag szükséges? Írassuk ki a termék nevét, a termék kódját, az előállításához szükséges anyagok azonosítóját, nevét és az előállításához szükséges mennyiséget!
2. Az elmúlt évben melyik megrendelésben szerepelt az „AA1” és „AA2” kódú termék együttesen? Írassuk ki rendelés dátumát, azon belül rendelési szám és termékkód szerint növekvő sorrendben rendelés dátumát, a rendelési számokat és megrendelt termék kódját és nevét!
3. Írassuk ki, hogy 2001. első félévében melyik megrendelő rendelt meg legalább 10 darab „AA1” kódú terméket és legalább 20 darab „AA2” kódú termék is!

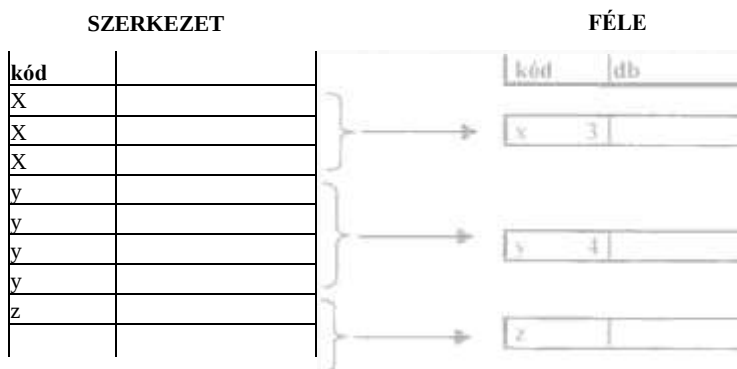
10. Maximumkeresés

Melyik termék áll a legtöbbféle anyagból?

(Figyeljünk a megfogalmazás pontatlanságára: termék helyett termékek értendő, mert több termék is állhat ugyanannyiféle anyagból, a kérdésre több válasz is lehetséges!)

1. Határozzuk meg, hogy hányféle anyagból állnak a termékek!

A SZERKEZET tábla soraiból csoportokat képezhetünk a kód mező értékei szerint. Maximum annyi csoport lehet, ahányféle termék van. Lehet kevesebb is, mivel elvileg lehet olyan termék is, amelyiknek nincs megadva a szerkezete. Egy csoporthoz annyi sor tartozik, ahányféle anyagból áll a termék.



```
SELECT kód, COUNT(*)ASdb FROM szerkezet GROUP BY kód;
```

-> FÉLEfkód, db}

SELECT mezőlistájában csak olyan információ lehet, ami az egész csoportra jellemző:

- a GROUP BY záradékban használt csoportosítási ismértv(-ek),
- aggregáló függvények (pl.: hány sor tartozik a csoporthoz),
- konstansok.

```
SELECT kód, COUNT(*) AS db,
       "féle anyagból áll" AS konstans
FROM szerkezet
GROUP BY kód
ORDER BY COUNT(*) DESC, kód;
```

•> FÉLE_3{kód, db, konstans}

Ennek a táblának már három oszlopa van: kód, db, konstans. A konstans értéke mindig „féle anyagból áll”. A tábla elején lesznek azok a termékek, ahol a db értéke a legnagyobb, azaz amelyik termékek a legtöbbféle anyagból állnak. Az azonos db értékű termékek kód növekvő sorrendben fognak megjelenni.

Az újabb SQL szabvány megengedi, hogy a SELECT utasítás mezőlistájában is lehessen beágyazott SELECT utasítás. (A belső lekérdezés ebben az esetben csak egyetlen mezőt adhat vissza.)

```
SELECT külső.kód, (SELECT COUNT(*)
                    FROM szerkezet belső
                    WHERE belső.kód = külső.kód) AS db
FROM szerkezet külső;
```

A külső lekérdezés kiválaszt egy sort a SZERKEZET táblából, és a sorhoz a belső lekérdezés meghatároz egy olyan értéket, amely egyenlő a kiválasztott sorban lévő kód mező értékével egyenlő kód értékeket tartalmazó sorok számával. A lekérdezésnek ugyanannyi sora lesz, mint az eredet SZERKEZET' táblának. Minden kód annyiszor fog előfordulni, ahányféle anyagból áll a termék. Azért, hogy ne legyenek azonos sorok az eredményben, használhatjuk a DISTINCT szót.

```
SELECT DISTINCT külső.kód, (SELECT COUNT(*)
                             FROM szerkezet belső
                             WHERE belső.kód = külső.kód) AS db
FROM szerkezet külső;
```

Az eredménytábla azonos lesz, mint az első lekérdezésben (FELE lekérdezés), csak sokkal bonyolultabban jutottunk el a megoldáshoz.

Több információt kapunk, ha külső lekérdezésben a SZERKEZET tábla helyett a TERMÉK táblából indulunk ki.

```
SELECT termék.kód, (SELECT COUNT(*)
                    FROM szerkezet
                    WHERE szerkezet.kód = termék.kód) AS db FROM
termék;
```

Ha a feladat olyan lett volna, hogy a cég nemcsak összeszerelt termékeket árusít. hanem kész termékek továbbadásával is foglalkozik, aminek csak az eladási árát tároljuk a rendszerben, akkor nem volna szerkezetleírás csak terméksor. Ebben az esetben, ellentétben az előző megoldással, ezek a termékek is megjelennének az eredmény táblában, nulla darabszámmal. Azt, hogy minden esetben kell egy terméknek szerkezetleírása, vagy sem, az adatbázis felállításánál megszorításokkal tudjuk előírni.

A DISTINCT szó ebben a lekérdezésben felesleges, hiszen a TERMÉK tábla sorait írtuk ki, ahol a termék kódja kulcs szerepű mező.

2. Keressük meg, hogy melyik termékek állnak legalább háromféle anyagból!

```
SELECT kód, COUNT(*)ASdb
      FROM szerkezet
      GROUP BY kód
      HAVING COUNT(*) >= 3;
```

Ebben a táblában a FÉLE táblának csak azoknak a sorai lesznek benne, amelyekben a db értéke nagyobb vagy egyenlő mit három. Azt nem tudhatjuk, mennyi legtöbb, csak próbálkozhatunk a hármas növelésével addig, amíg a táblázat üres lesz, és akkor a maximum a még nem üres táblázathoz tartozó szám.

Készíthetünk egy paraméteres lekérdezést is erre a célra:

```
SELECT kód, COUNT(*)ASdb
      FROM szerkezet
      GROUP BY kód HAVING COUNT(*) >= [féle termékből áll];
```

Remélem soha senkinek sem fog eszébe jutni, hogy így keresse meg a legnagyobb értéket!

3. Határozzuk meg, hogy maximálisan hányféle anyagból állnak a termékek!

```
SELECT MAX(db) AS mdb
      FROM féle;
```

LEGTÖBB{mdb}

LEGTÖBB

mdb

x

A táblázatnak egy oszlopa és egy sora van.

írassuk ki a maximum mellé itt is az előzőekben kiírt „féle anyagból áll” konstanst, ami konstans mezőnévvel benne van a FÉLE 3 táblában is.

```
SELECT MAX(db) AS mdb, konstans
      FROM féle_3;
```

Ez így nem jó!

A fÉLE3 konstans nevű mezőjéről csak én tudom, hogy az egész táblában ugyanaz az értéke, azaz jellemző az egész táblára. Az SQL nem foglalkozik a mező értékével, az eredmény biztosan egy sor lesz, a konstans nevű mező értéke pedig akármilyen lehet.

```
SELECT MAX(db) AS mdb, "féle anyagból áll" AS konstans2 FROM
    féle_3;
```

így már jó, egy sora és két oszlopa van az eredmény táblának.

mdb	konstans2
1 x	féle anyagból áll

Megoldhatjuk úgy is a feladatot, hogy a FÉLE3 táblából írjuk ki a konstans2-t, de akkor abból az oszlopból is csak egy értéket írathatunk ki, vagy a maximumot, vagy a minimumot. Jelen esetben mindegy, hogy melyiket, tekintve, hogy a konstans nevű mezőnek minden értéke egyenlő egymással.

```
SELECT MAX(db) AS mdb, MAX(konstans2) AS konstans2 FROM
    féle_3;
```

- Keressük meg, hogy melyek azok a termékek, amelyek a legtöbbnek anyagból állnak! (Csak a termékódot kell kiírni.)

. Megoldás

Meg kell keresni a FÉLE táblában azokat a kódokat, amelyekhez a LEGTÖBB JHában lévő mdb érték tartozik.

```
SELECT kód, db
    FROM féle, legtöbb WHERE db = mdb;
```

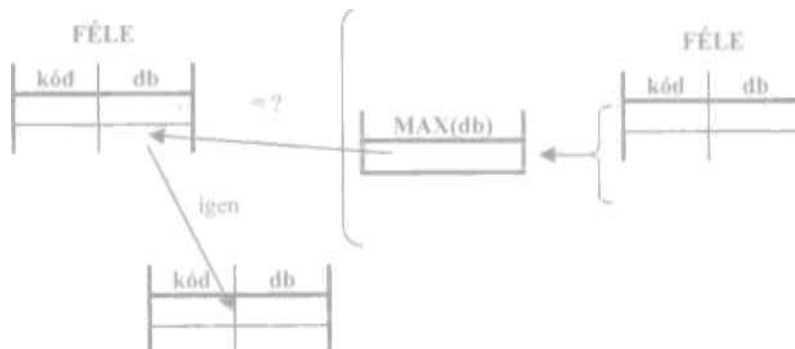
FÉLE és a LEGTÖBB tábla minden sorának minden sorával való párosítása - ez annyi sort jelent, mint amennyi a FÉLE táblázatnak volt, mert a LEGTÖBB -nak pontosan egy sora volt) csak azt, illetve azokat fogja kiírni, ahol a db egyenlő mdb-vel, azaz, amelyek a legtöbbféle anyagból áll, illetve állnak.

' megoldás

■ 3. kérdés és a 4. kérdés I. megoldásának az egyesítésével:

```
SELECT kód, db
    FROM féle
    WHERE db = (SELECT MAX(db)
                FROM féle);
```

A belső SELECT utasítás eredménye biztosan egyetlen értéket ad vissza, ezért használhatunk = jelet.



III Megoldás

Az 1. lekérdezésbe beépítve a 4. lekérdezés II. megoldását a HAVING záradék segítségével:

```
SELECT kód, COUNT(*)ASdb
FROM szerkezet
GROUP BY kód HAVING COUNT(*) = (SELECT MAX(db) FROM
féle);
```

Ez a megoldás kevésbé hatékony, mint az előző, és nem ad több információt sem. A FÉLE táblára itt is szükség van, tehát a SZERKEZET táblát itt is kétszer dolgoztuk fel. Érdeemes megfigyelni azonban, hogy a WHERE záradék feltétele átkerült a HAVING záradék feltételébe.

IV Megoldás

Az SQL 92 szabvány megengedi, hogy a FROM záradékban a táblanév helyett is használhassunk SELECT utasítást (ACCESS 2000 már megengedi, az előző ACCESS még nem engedte meg). Ebben az esetben az SQL összeállít a belső SELECT utasítás alapján egy ideiglenes táblát, amelyet betesz a FROM záradékba, és utána hajtja végre a külső SELECT utasítást.

```
SELECT kód, COUNT(*)ASdb
FROM szerkezet
GROUP BY kód HAVING COUNT(*) = (SELECT MAX(db)
FROM (SELECT kód, COUNT(*)ASdb
FROM szerkezet
GROUP BY kód));
```

vagy:

```
SELECT kód, db
FROM (SELECT kód, COUNT(*) AS db
      FROM szerkezet
      GROUP BY kód)
WHERE db =
      (SELECT MAX(db)
       FROM (SELECT COUNT(*) AS db
             FROM szerkezet GROUP BY kód));
```

Vegyük észre, hogy sehol sem volt szükségünk a mezőnevek minősítésére, mindenhol egyértelmű, hogy melyik mezőnév melyik táblához tartozik. Természetesen mindenhol használhattunk volna hivatkozásokat a mezőnevek előtt:

```
SELECT táblai.kód, táblai.db
FROM (SELECT belsői.kód, COUNT(*) AS db
      FROM szerkezet belsői
      GROUP BY belsői.kód) táblai
WHERE táblai.db =
      (SELECT MAX(tábla2.db)
       FROM (SELECT COUNT(*) AS db FROM
             szerkezet belsői2 GROUP BY
             belsői2.kód) tábla2);
```

Az utasítás jó (inkább szép), de elég nehezen átlátható. Kerüljük az ilyen bo- olultságú megoldásokat még akkor is, ha az SQL megengedi! Mindig gondolunk arra, hogy nem elég egy szép utasítást kitalálni, másnap, vagy egy év múl- a meg kell érteni azt nekünk és másoknak is!

Sí A legtöbbféle anyagból álló termékeknek nemcsak a kódját, ha- " cm a nevét is írassuk ki!

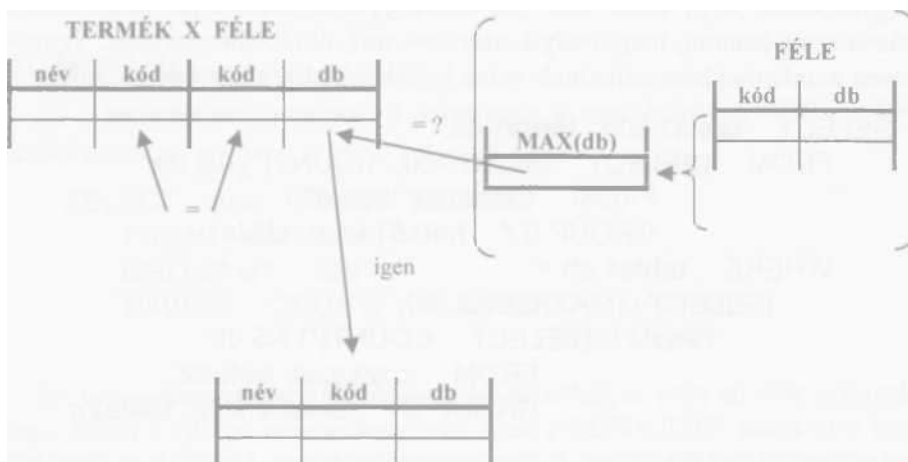
Ha a legtöbbféle anyagból álló termék nevét is szeretnénk megtudni és nem- .sak a termék kódját, akkor kell a TERMÉK tábla is!

Nem biztos, hogy a név mező különböző értékeket tartalmaz, mert a név nem dics a TERMÉK táblában, ezért a kódot is kiírjuk. (Az adatbázis-kezelőkben -egszorításként megadhatjuk, hogy a név mező egyedi legyen. Az ACCESS-ben úgy adhatjuk meg, hogy a név mező indexelt legyen, és nem tartalmazhat azo- értékeket, de ezt most nem használjuk ki.)

I. Megoldás

írassuk ki a termék nevét, kódját és azt, hogy hányféle anyagból áll. A termékkód kulcs a TERMÉK táblában, ezért a kiíratásával minden sor különböző lesz az eredménytáblában.

```
SELECT név, termék.kód, db
FROM termék, féle
WHERE termék.kód = féle.kód
AND db = (SELECT MAX(db)
          FROM féle);
```



II. Megoldás

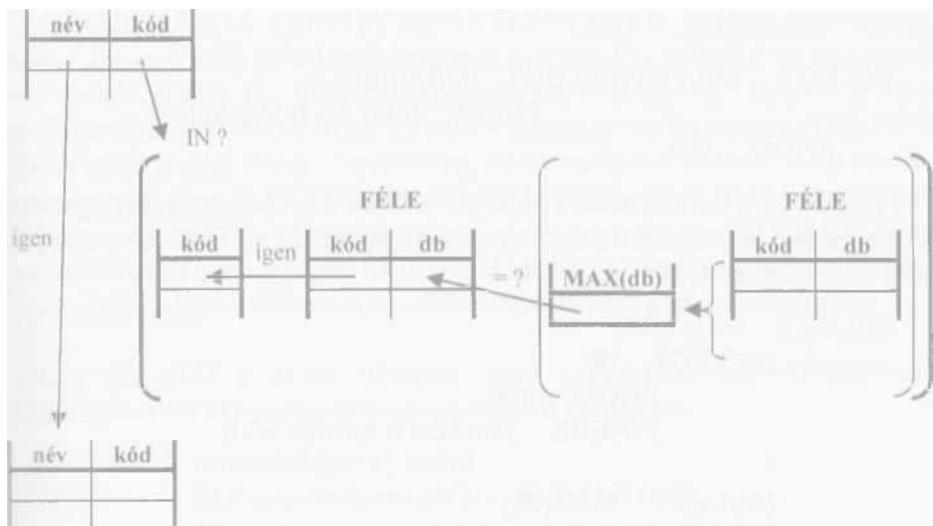
Egyszerűsítsük a feladatot, csak azoknak termékeknek a nevét és kódját szeretnénk kiíratni, amelyek a legtöbbféle anyagból állnak, és nem kell az, hogy hányféle anyagból állnak.

Ebben az esetben elég a TERMÉK tábla sorain végigmenni, és eldönteni, hogy a termék kódja megfelel-e a 4. lekérdezés II. megoldásában megfogalmazottaknak.

```
SELECT név, kód
FROM termék
WHERE kód IN
  (SELECT kód
   FROM féle
   WHERE db = (SELECT MAX(db)
               FROM féle));
```

A lényeges különbség a két megoldás között az, hogy a második megoldásban a kiírás csak a TERMÉK táblából történik, a FÉLE táblából a db kiíratási céllal nem érhető el. A db értékét csak a két belső SELECT utasításban használhatjuk. Ezzel szemben az első megoldásban a kiírás a TERMÉK és a FÉLE tábla direktszoratából indul ki, így a db értéke is kiíratható.

TERMÉK



Figyeljük meg a középső SELECT utasítás és a legbelső SELECT utasítás közötti különbséget:

- a középső SELECT utasítás több értéket adhat vissza, ezért értékei között IN predikátummal válogattunk,
- a legbelső SELECT utasítás mindig egy értéket ad vissza, ezért értékét = jellel hasonlítottuk össze.

- Írassuk ki a termék kódját és azt az információt, hogy az adott termékben lévő anyagfélések darabszáma hogyan viszonylik a legbelső anyagfélést tartalmazó termék anyagféléseének a darabszámához! (A termék bonyolultságáról szeretnénk információt kapni.)

Térjünk vissza a 4. feladat I. megoldásához. A FÉLE táblázat minden sorát kell raktári felhasználni a LEGTÖBB táblázatban.

```
SELECT kód, db/mb AS bonyolultság
FROM féle, legtöbb;
```

Vegyük észre, hogy a lekérdezésben két táblázatot használtunk fel, és nincs összekapcsoló feltétel. Ezt azért tehettük meg, mert a LÉGI ÖBB táblának csak egy sora van, így a direktszorzatnak ugyanannyi sora lesz, mint a FELE táblának.

Ha a LEGTÖBB tábla helyett a 4. feladat II. megoldása szerinti beágyazott SELECT utasítást szeretnénk használni, a külső SELECT utasítás mezőlistájában nem használhatjuk a belső SELECT által visszaadott értéket. A maximumot kiszámító beágyazott SELECT utasítást a külső SELECT utasítás mezőlistájába kell írni.

```
SELECT kód, db/(SELECT MAX(db)
                  FROM féle) AS bonyolultság
FROM féle;
```

Végezetül, ha a termék nevét és kódját valamint a bonyolultságát is egyetlen lekérdezésben szeretnénk kiírni, és csak a TERMÉK és a FÉLE táblát akarjuk hozzá felhasználni:

```
SELECT név, kód,
       (SELECT db
        FROM féle
        WHERE féle.kód = termék.kód)
      /
      (SELECT MAX(db)
       FROM féle) AS bonyolultság
FROM termék;
```

Bonyolultság és név szerint rendezni az eredménytáblát ACCESS-ben csak úgy tudjuk, ha eltesszük előbb ezt a lekérdezést (az SQL szabvány szerint lehetne ezt a lekérdezést is rendezni, de az ACCESS nem engedi a beágyazott SELECT szerinti rendezést):

```
NÉV_BONY{név, kód, bonyolultság}
```

Majd ezt rendezzük bonyolultság és név szerint.

```
SELECT *
FROM név_bony ORDER BY bonyolultság, név, kód;
```

Gyakorló feladatok

1. Melyik megrendelésben rendelték meg a legtöbb „AA1” kódú terméket?
2. Melyik terméknek a legkisebb az anyagára?
3. Mikor rendelt meg utoljára „A” betűvel kezdődő partnerkódú megrendelő?

11. Csoportonkénti maximum keresése

1. A 2001. évi megrendeléseknek rendelésenként számítsuk ki a rendelés teljes értékét, és az eredménytáblát egészítsük ki a megrendelés hónapjának a sorszámaival!

A RENDELÉSFEJ, a RENDELÉS és a TERMÉK tábla „értelmes összekapcsolásában” (kulcs - külső kulcs) rendelésszáma csoportosítva a darab és ár szorzata adja egy rendelés értékét. Egy megrendeléshez csak egy rendelésdátum tartozik, de a létrejövő összekapcsolt tábla táblában egy rendeléshez annyi sor fog tartozni, ahányféle terméket megrendeltek ebben a rendelésben. Az összekapcsolt táblának annyi sora lesz, mint amennyi sora van RENDELÉS táblának. Az összekapcsolt táblát kell rendelésenként csoportosítani. Ha ki akaijuk írni a rendelés dátumát is, a GROUP BY záradék használata miatt csak aggregáló függvénnyel írhatunk ki nem csoportosító mezőnevet, ezért akár maximum, akár minimum használata kötelező a dátum kiírásánál.

Egy dátumból a dátum hónapját, mint számértéket az ACCESS-ben a Month(dátumérték) mezőfüggvény használatával kaphatunk.

```
SELECT rendelésfej.rend_száma,  
       SUM(rendelés.darab*termék.ár) AS érték,  
       MAX(Month(rendelésfej.rend_dátum)) AS hónap  
FROM rendelésfej, rendelés, termék  
WHERE rendelésfej.rend_száma = rendelés.rend_száma  
      AND rendelés.kód = termék.kód  
      AND rendelésfej.rend_dátum BETWEEN  
                                     #1/1/2001# AND #12/31/2001#  
GROUP BY rendelésfej.rend_száma;  
        -> REND_HÓNAP{rend_száma, érték, hónap}
```

2. Válasszuk ki a 2001. év legnagyobb megrendeléseit!

Az éves legnagyobb érték a REND_HÓNAP táblát felhasználva:

```
SELECT MAX(érték) AS legnagyobb  
FROM rend_hónap;  
        -> ÉVES_LEG{legnagyobb}
```

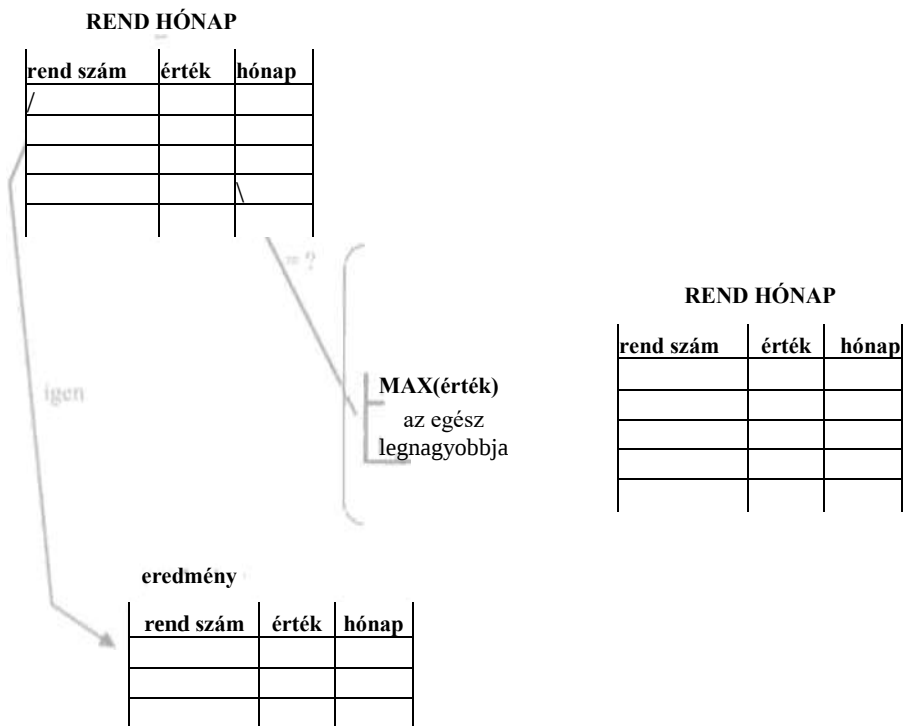
Az éves legnagyobb értékű rendelések:

```
SELECT rend_száma, érték AS [évi legnagyobb]
FROM rend_hónap, évesjeg
WHERE érték = legnagyobb;
```

A két lekérdezés összevonva egy lekérdezéssé:

```
SELECT rend_száma, érték AS [évi legnagyobb]
FROM rend_hónap
WHERE érték = (SELECT MAX(érték)
               FROM rendhónap)
ORDER BY rend_száma;
```

A külső SELECT utasítás végigmegy a REND HÓNAP tábla minden során, és azokat a sorokat fogja elfogadni, amelyeknek az értéke egyenlő a belső SELECT utasításban meghatározott értékkel. A belső SELECT utasítás által visszaadott érték független a külső SELECT utasításban éppen vizsgált sor értékétől, mindig az összes megrendeléseknek a maximális értékét adja vissza, ezért egyszer, a külső SELECT utasítás előtt, értékelődik ki.



3. Havonként csoportosítással válasszuk ki a 2001. évi megrendelésekből a havi legnagyobb értékű megrendeléseket!

Lényegében az előző feladatnak megfelelő szerkezetet felhasználva, számíthatjuk ki a havi legnagyobb értékű megrendeléseket.

Havonként a legnagyobb értékek (egy érték havonta):

```
SELECT MAX(érték) AS legnagyobb, hónap
FROM rend_hónap
GROUP BY hónap;

HAVI_LEG{legnagyobb, hónap}
```

Havonként a legnagyobb értékű rendelések (havonta több megrendelés is lehet):

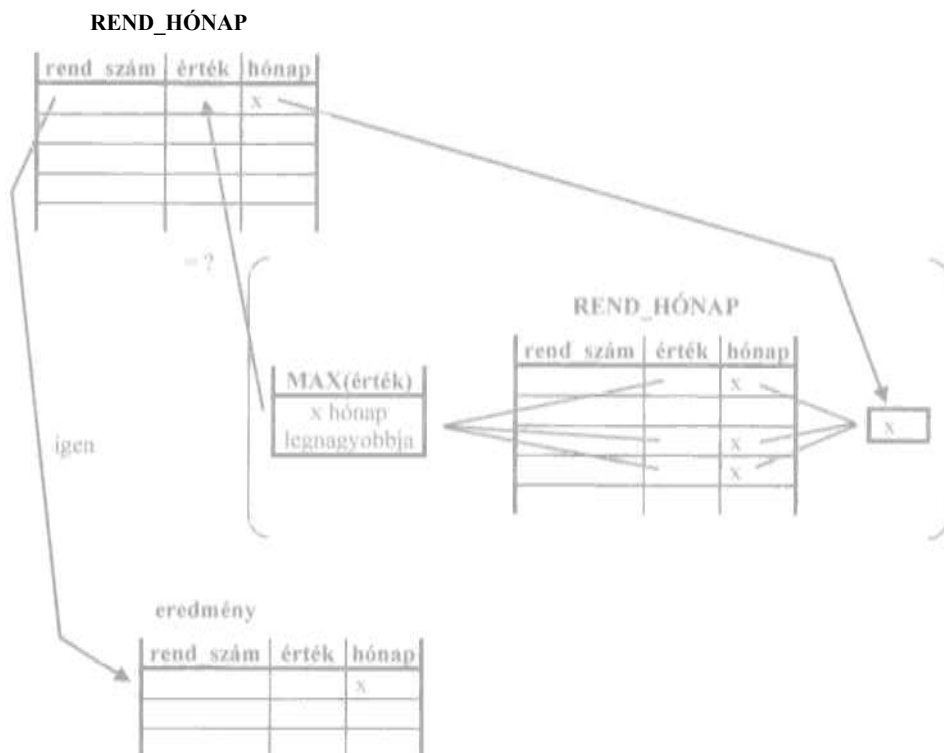
```
SELECT rend_száma, érték AS [havi legnagyobb], rend_hónap.hónap
FROM rend_hónap, havijeg
WHERE rend_hónap.hónap = havi_leg.hónap
AND érték = legnagyobb;
```

A REND HÓNAP táblának azok a sorai fognak megfelelni a feltételnek, ahol ■ 'dkét táblában a hónap megegyezik és a saját hónapjának megfelelő legnagyobb értékű megrendelés.

Az előző két lekérdezés összevonva:

```
SELECT rend_száma, érték AS [havi legnagyobb], hónap FROM
rend_hónap külső
WHERE érték = (SELECT MAX(érték)
FROM ver\ő_Y\ór\ap 'oe\ső
WHERE belső.hónap = külső.hónap)
ORDER BY hónap, rend_száma;
```

külső SELECT utasítás végigmegy a REND HÓNAP tábla minden során, és • a sorokat fogja elfogadni, amelyeknek az értéke egyenlő a belső SELECT utasítás meghatározott értékkel. A belső SELECT utasítás - ellentétben az előző pontéi - egfogalmazottakkal - nem független a külső SELECT utasításban éppen vizs- _ fenol, azoknak a megrendeléseknek a maximális értékét adja meg, amelyek _ ir _ hhoz a hónaphoz tartoznak, mint a külső SELECT utasításban található meg-



Gyakorló feladatok

1. Havonta melyik megrendelésben rendelték meg a legtöbb „AA 1” kódú terméket?
2. Az anyag nevének első betűje szerint melyik terméknek a legkisebb az anyagára?
3. Melyik partner mikor rendelt utoljára?

12. Adatbázisbeli megszorítások ellenőrzése

1. Határozzuk meg, hogy a termékek hányféle anyagból épülnek fel! írjuk ki a termék nevét, valamint azt, hogy a termék hányféle anyagból épül fel!

Első lépésben írjuk ki a termék kódját és azt, hogy a termék hányféle anyagból épül fel. A feladat megoldásához elég a SZERKEZET tábla.

```
SELECT kód, COUNT(*)  
FROM szerkezet  
GROUP BY kód;
```

Ha a termék nevét akarjuk kiírni, szükség lesz a TERMEK táblára is. A TERMEK táblánál nem tettünk olyan megszorítást, hogy a név egyedi legyen. A termékkód csak egyszer szerepelhet a táblában, mivel a tábla kulcsa a kód, ami azt elenti, hogy a kód egyértelműen meghatározza a név értékét, de általánosságban ez fordítva nem igaz. Egyáltalán nem biztos, hogy minden feladatnál igaznak kell ennie, hogy a név egyedi. Azt, hogy egy név egyedi legyen vagy sem, nem adatbázis tervezésénél kell eldönteni, hanem a valóságot kell megvizsgálni, és az igényeknek megfelelően kell az adatbázist megtervezni.

Ha a TERMÉK tábla felépítésénél a név egyediségét megkötöttük volna, akkor a GROUP BY záradékba beírhatnánk a termék nevét. Ebben az esetben nem tesz az eredmény táblának se kevesebb, se több sora, mint az előző lekérdezésben, és a COUNT(*) értéke is ugyanaz marad. (Több semmiképpen nem lehet, mert a termékkódok száma nem lehet több mint, a terméknevek száma.)

```
SELECT név, COUNT(*)  
FROM termék, szerkezet WHERE termék.kód = szerkezet.kód  
GROUP BY név;
```

Példánkban, amikor a név egyediségét nem kötöttük ki, a kód szerinti csoportosítás adja meg azt, hogy minden termékről szóló információ külön sorban szerepeljen, ezért kódra kell csoportosítani. Ha a termék nevét is ki akarjuk írni, Ák ar a termék nevét, mint a csoport tulajdonságát kell kiírni.

```
SELECT termék.kód, MAX(név), COUNT(*)  
FROM termék, szerkezet  
WHERE termék.Kód = szerkezet.kód  
GROUP BY termék.kód;
```

2. Keressük meg azokat a termékneveket, amelyek nem egyediek!

Határozzuk meg a TERMÉK táblában a különböző nevek számát, a különböző kódok számát és a tábla sorainak a számát.

```
SELECT COUNT(DISTINCT név)
           AS [a különböző nevek száma],
       COUNT(DISTINCT kód)
           AS [a különböző kódok száma],
       COUNT(*) AS [a sorok száma]
FROM termék;
```

A két utolsó szám, mindig egyenlő egymással, a kód kulcs a TERMÉK táblában, és ebből következik, nem lehet két különböző sorban azonos kód. Ebben a táblában a COUNT(DISTINCT kód) és a COUNT(kód) mindig egyenlő egymással. Ha az adatbázisban minden terméknév egyedi, és mindenhol ki van töltve, akkor mindhárom érték egyenlő.

A COUNT(DISTINCT mezőnév) kifejezés az ACCESS 2000-ben nem működik, bár standard SQL utasítás. ACCESS 2000-ben így írhatjuk a fenti utasítást:

```
SELECT DISTINCT név AS t_név
FROM termék;
                                -» TERMÉKNÉV{t_név}
```

```
SELECT COUNT(t_név) AS [a nevek száma]
FROM terméknév;
                                -> NÉVSZÁM{a nevek száma}
```

```
SELECT MIN([a nevek száma]) AS [a különböző nevek száma],
       COUNT(kód) AS [a különböző kódok száma], COUNT(*) AS
       [a sorok száma]
FROM termék, névszám;
```

A TERMÉK és a NÉVSZÁM táblából készített direktszorzatban a nevek száma minden sorban ugyanaz az érték, de kiíratni, mivel az eredménytábla csak egy sorból áll, csak egyetlen értéket lehet. Ezért írtuk a SELECT mezőlistájában a MIN([a nevek száma]) kifejezést.

Ha az első és harmadik érték nem egyenlő egymással, azaz a különböző nevek száma és a táblázatban lévő sorok száma nem egyenlő, írassuk ki azokat a neveket, amelyekhez több kód tartozik.

```
SELECT név, COUNT(*) AS [ennyiszer fordul elő ez a név] FROM
termék GROUP BY név
HAVING COUNT(*)>1;
```

Ha ez a tábla üres, akkor minden név egyedi.

3. írassuk ki TERMÉK táblából azokat a kódokat, amelyek nem találhatók meg a SZERKEZET táblában, azaz azokat, amelyeknek nem adtuk meg a felépítését!

A TERMÉK táblában lévő kódokat vizsgáljuk meg, hogy benne vannak-e a SZERKEZET táblából kigyűjtött kódok között.

```
SELECT kód AS [nincs szerkezetei]
FROM termék
WHERE kód NŐT IN (SELECT kód
                  FROM szerkezet);
```

Ha az eredménytábla üres, akkor minden terméknek megadtuk a szerkezetét is.

A TERMÉK és a SZERKEZET tábla szerepét felcserélve megkaphatjuk azokat a felesleges szerkezetleírásokat, amelyek nem tartoznak valódi termékekhez.

```
SELECT kód AS [nincs hozzá termék]
FROM szerkezet
WHERE kód NŐT IN (SELECT kód
                  FROM termék);
```

Ez a feladat leginkább akkor fordulhat elő, amikor az adatbázist külső fájlokból szeretnénk feltölteni. Az adatok betöltése előtt célszerű megvizsgálni, hogy a betöltendő adatok jók-e, teljesítik-e az integritási feltételeket. Később, amikor már használjuk az adatbázist, az ilyen jellegű feltételeket megszorításként megadhatjuk, így a nem megfelelő adatok nem is kerülhetnek be az adatbázisba.

Gyakorló feladatok

1. Ellenőrizzük, hogy minden SZERKEZET táblabeli azonosító előfordul-e a TERMÉK táblában!
2. Keressük meg, hogy melyek azok a terméknevek, amelyek nem egyediek!
3. Keressük ki a RENDELÉS táblából azokat a sorokat, amelyekhez nem tartozik a RENDELÉSFEJ táblában sor. írassuk ki az ilyen sorok minden adatát, majd készítsünk egy listát, amelyik tartalmazza minden rossz rendelésszámhoz a RENDELÉSFEJ táblából azokat a rendelésszámokat, amelyek csak az első karakterben különböznek az adott rossz rendelésszámtól! (Feltételezzük, hogy az első karaktert véletlenül elírták.)

13. Két szempont szerinti csoportosítás

1. Gyűjtsük ki az adatbázisból, hogy melyik partner milyen termékből mennyit rendelt meg!

I. Megoldás

A rend szám alapján összekapcsolt RENDELÉSFEJ és RENDELÉS táblát először a partner kódja, majd a termék kódja szerint csoportosítsuk.

```
SELECT partner_kód, kód, SUM(darab)
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_szám = rendelés.rend_szám GROUP
BY partner_kód, kód;
```

II. Megoldás

Készítsük el ugyanezt a lekérdezést, de fordított csoportosítással.

```
SELECT kód, partner_kód, SUM(darab)
FROM rendelésfej, rendelés WHERE rendelésfej.rend_szám =
rendelés.rend_szám GROUP BY kód, partner kód;
```

A két lekérdezésnek ugyanannyi sora van, a sorok tartalma ugyanaz, csak a sorok rendezettsége eltérő. Az eltérést nem a mezőlista eltérő sorrendje határozza meg, hanem a csoportosításban lévő szempontok sorrendje. Az első lekérdezésben a sorok partner kód és azon belül kód, a másodikban kód és azon belül partner_kód sorrendben fognak megjelenni.

Azért, hogy ezt világosan megértsük, nézzünk egy példát!

Legyenek a partnerek által megrendelt termékek kódjai (3 partner, 2 termék), azaz az összetartozó partnerkódok és termékkódok:

{aab; AA1}, {aab; AA1}, {aab; AA2}, {aac; AA1},
{aac; AA2}, {aac; AA2}, {aad; AA1}.

Partnerkód szerint csoportosítva három csoportot kapunk:

1. szempont	aab	aac	aad
a csoportba tartozó elemek	{aab;AA1} {aab; AA1} {aab; AA2}	{aac; AA1} {aac; AA2} {aac; AA2}	{aad; AA1}

Partnerkódon belül termékkód szerint csoportosítva hat csoportot kaphatnánk, de egy csoport üres lesz:

1. szempont	aab		aac		aad	
2. szempont	AA1	AA2	AA1	AA2	AA1	AA2
a csoportba tartozó elemek	{aab; AA1} {aab; AA1}	{aab; AA2}	{aac; AA1}	{aac; AA2} {aac; AA2}	{aad; AA1}	

Termékkód szerint csoportosítva két csoportot kapunk:

1. szempont	AA1	AA2
a csoportba tartozó elemek	{aab; AA1} {aab; AM} {aac; AA1} {aad; AA1}	{aab; AA2} {aac; AA2} {aac; AA2}

Termékkódon belül partnerkód szerint csoportosítva szintén hat csoportot kaphatnánk, de egy csoport itt is üres lesz:

1. szempont	AA1			AA2		
2. szempont	aab	aac	aad	aab	aac	aad
a csoportba tartozó elemek	{aab; AA1} {aab; AA1}	{aac; AA1}	{aad; AA1}	{aab; AA2 }	{aac; AA2} {aac; AA2}	

Mindkét esetben egy csoportba azok az elemek (táblasorok) kerülnek, amelyek mindkét szempontnak megfelelnek. Ebből az is következik, hogy a csoport elemeiből számított bármilyen érték is (maximum, minimum stb.), az egymásnak megfelelő csoportokban, csoportosítás sorrendjétől függetlenül, ugyanolyan értékű lesz.

Az eredménytábla sorainak a sorrendjében lesz csak különbség, tekintve, hogy az SQL a GROUP BY záradékban felsorolt csoportosító mezők szerint, növekvő sorrendben jeleníti meg a sorokat.

Példánkban ez azt jelenti, hogy a fenti két SQL utasításnak az eredménytáblája mindkét esetben öt soros lesz. Az eredménytábla sorait a FROM és a WHERE záradékkal meghatározott táblából úgy kapjuk meg, hogy a fenti táblázatok utolsó sorában lévő cellákat balról jobbra megfeleltetjük az eredménytábla sorainak, és az így kapott csoportból számítjuk ki a SUM(darab) értékét. Üres csoportból az SQL nem készít eredménysort.

Gyakorló feladatok

1. Havonként és megrendelőnként írassuk ki a 2001. évi megrendelések összértékét!
2. Készítsünk egy lekérdezést, amely havi bontásban megmutatja, hogy melyik partner hány esetben adott valamilyen megrendelést.
3. Hét napra előre naponként írassuk ki, hogy az elkészítendő termékek előállításához milyen anyagból mennyire lesz szükség! A már elkészült termékek anyagszükségletét ne vegyük figyelembe.

14. Az eredménytáblát szűkítő feltételek

1. Gyűjtsük ki az adatbázisból, hogy melyik partner milyen termékből mennyit rendelt meg 2001. januárjában!

A megrendelt teljes állományra a termékek darabszámát a rend szám alapján összekapcsolt RENDELÉSFEJ és RENDELÉS táblából először a partner kódja, majd a termék kódja szerint csoportosítással kaphatjuk meg.

```
SELECT partner_kód, kód, SUM(darab)
FROM rendelésfej, rendelés WHERE rendelésfej.rend_száma =
rendelés.rend_száma GROUP BY partner_kód, kód;
```

A rendelés dátuma nem szerepel az eredménytáblában, de a felhasznált RENDELÉSFEJ táblában szerepel, ezért a szűkítő (kiválasztó) feltételt a WHERE záradékba kell beírni.

```
SELECT partner_kód, kód, SUM(darab)
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_száma = rendelés.rend_száma
AND rend_dátum BETWEEN
#01/01/2001# AND #01/31/2001#
GROUP BY partner_kód, kód;
```

2. Szűkítsük tovább az eredménytáblát, hogy csak azok a termékek kerüljenek be az összesítésbe, amelyekből az adott megrendelő több mint 500 darabot rendelt meg 2001. januárjában!

Sem a RENDELÉSFEJ, sem a RENDELÉS táblában nincs benne, hogy egy partner egy termékből 2001. januárban összesen mennyit rendelt meg, így a WHERE záradékban erre vonatkozó feltételt nem adhatunk meg. Partnerenként és termékenként a rendelt darabszám az eredménylistában van, ezért HAVING záradék segítségével szűkíthetjük tovább a listát.

```
SELECT partner kód, kód, SUM(darab)
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_száma = rendelés.rend_száma AND
rend_dátum BETWEEN
#01/01/2001# AND #01/31/2001#
GROUP BY partner_kód, kód
HAVING SUM(darab) > 500;
```

3. Szűkítsük tovább az eredménytáblát, hogy csak azok a termékek kerüljenek be, amelyek kódja „A” betűvel kezdődik, és amelyekből az adott megrendelő több mint 500 darabot rendelt meg 2001. januárjában!

A termék kódja benne van egyrészt a RENDELÉS táblában, másrészt az eredménytáblában is, ezért választhatunk, a szűkítést megtehetjük vagy a WHERE záradék, vagy a HAVING záradék segítségével.

```
SELECT partner_kód, kód, SUM(darab)
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_szám = rendelés.rend_szám
AND rend_dátum BETWEEN
#01/01/2001# AND #01/31/2001#
AND kód LIKE "A*"
GROUP BY partner_kód, kód HAVING SUM(darab) > 500;
```

vagy:

```
SELECT partner_kód, kód, SUM(darab)
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_szám = rendelés.rend_szám AND
rend_dátum BETWEEN
#01/01/2001# AND #01/31/2001#
GROUP BY partner kód, kód
HAVING SUM(darab) > 500
AND kód LIKE "A*";
```

Mindkét lekérdezés ugyanazt az eredményt adja. A két lekérdezés között az a különbség, hogy az első esetben már a csoportosítás előtt elhagytuk azokat a sorokat, amelyek nem „A” betűvel kezdődtek, a második lekérdezésben ezekre is elvégeztük a csoportosítást. Az első lekérdezés ezért gyorsabban fog végrehajtódni.

4. Szűkítsük még tovább az eredménytáblát, hogy csak azok a termékek kerüljenek be az összesítésbe, amelyekből összesen legalább 1000 darabot rendeltek meg 2001. januárjában, és amelyek kódja „A” betűvel kezdődik, és amelyekből egy adott megrendelő több mint 500 darabot rendelt meg 2001. januárjában!

Termékenként a 2001. januári összes megrendelés sem a RENDELÉSFEJ, sem a RENDELÉS táblában, sem az eredménytáblában nincs benne, ezért az eredménytábla további szűkítéséhez ezeket az értékeket ki kell számítani, amit egy belső SELECT utasítással tehetünk meg.

```
SELECT partner_kód, kód, SUM(darab)
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_száma = rendelés.rend_száma
AND rend_dátum BETWEEN
                                #01/01/2001# AND #01/31/2001#
AND kód LIKE "A*"
AND kód IN (SELECT kód
                                FROM rendelésfej, rendelés WHERE
                                rendelésfej.rend_száma = rendelés.rend_száma AND
                                rend_dátum BETWEEN #01/01/2001# AND #01/31/2001#
                                GROUP BY kód
                                HAVING SUM(darab) >1000)
GROUP BY partner_kód, kód HAVING SUM(darab) > 500;
```

Az időre vonatkozó korlátozásra mind a külső, mind a belső SELECT utasításban szükség van. Ha a belső SELECT utasításba betesszük, hogy ellenőrizze, és csak az „A” betűvel kezdődőkkel foglalkozzon, akkor a belső SELECT utasítás kevesebb sort fog csoportosítani, gyorsabb lesz. Ekkor a külső SELECT utasításból elhagyható ez a feltétel, mert az IN feltétel már csak az „A” betűvel kezdő- dőkre lesz igaz.

```
SELECT partner_kód, kód, SUM(darab)
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_száma = rendelés.rend_száma
AND rend_dátum BETWEEN
                                #01/01/2001# AND #01/31/2001#
AND kód IN (SELECT kód
                                FROM rendelésfej, rendelés
                                WHERE rendelésfej.rend_száma =
                                rendelés.rend_száma AND rend_dátum
                                BETWEEN #01/01/2001# AND
                                #01/31/2001# AND kód LIKE "A*"
                                GROUP BY kód
                                HAVING SUM(darab) >1000)
GROUP BY partner_kód, kód
HAVING SUM(darab) > 500;
```

Egy gyakorlati jó tanács: készítsük el először minden szűkítő feltétel nélkül a lekérdezést, majd minden szűkítő feltételt külön-külön megvizsgálva, döntsük el, hogy hogyan és hova tudjuk beépíteni.

Gyakorló feladatok

1. Keressük meg azokat a termékeket, amelyek anyagára nagyobb, mint 1000 Ft, és nincs bennük „a” betűvel kezdődő nevű anyag!
2. Listázzuk ki partnerkódonként összegezve azoknak a termékeknek a kódját és nevét, ami még nem készült el, és lejárt már az elkészítésére vállalt határidő!
3. Listázzuk ki, hogy az elmúlt évi megrendelésekhez havonta anyagonként mennyi volt az „a” betűvel kezdődő nevű anyagokból a felhasználás! A lista csak azokat az anyagokat tartalmazza, amelyek fontosabbak, azaz a felhasználásuk 10000 Ft felett volt.

is-
es
tás
ás-
dő-

OOU

15. N : M viszonyú táblák összekapcsolása

1. Határozzuk meg, hogy mennyi a 2001. november 10.-re vállalt megrendelések összes anyagszükséglete! (Anyagonként ki kell írni a szükséges anyag azonosítóját, nevét, az anyagból szükséges összes mennyiséget és az anyag mértékegységét!)

Számítsuk ki először a RENDELÉS táblából a megadott határidőre vállalt termékek darabszámát.

```
SELECT kód, SUM(darab) AS ossz
FROM rendelés
WHERE dátum = #11/10/2001#
GROUP BY kód;
```

ÖSSZDARAB{kód, ossz}

Az ÖSSZDARAB táblához vegyük hozzá a SZERKEZET és az ANYAG táblát, és csoportosítsuk az anyag azonosítójára. A három tábla közötti kapcsolat:

- az ÖSSZDARAB és a SZERKEZET tábla között 1 : N,
- a SZERKEZET és az ANYAG tábla között N : 1 kapcsolat van.

```
SELECT anyag.azonosító, MAX(anyag.neve) AS anyagnév,
       SUM(összdarab.össz * szerkezet.mennyiség)
       AS [összes anyag], MIN(anyag.mért_egys) AS
       mértékegység
FROM összdarab, szerkezet, anyag
WHERE összdarab.kód = szerkezet.kód
      AND szerkezet.azonosító = anyag.azonosító GROUP BY
anyag.azonosító;
```

A két lekérdezést összevonhatjuk egy lekérdezéssé. Az ÖSSZDARAB tábla helyett a RENDELÉS tábla fog szerepelni a lekérdezésben.

A kapcsolatok:

- a RENDELÉS és a SZERKEZET tábla között nincs kapcsolat, nincs közöttük kulcs - külső kulcs kapcsolat (a valóságban, nem a relációs adatmodell- ben, N : M viszony van a kétfajta egyed között), de van mindkettőben olyan mező, amelyik a termékkódokat tartalmazza,
- a SZERKEZET és az ANYAG tábla között N : 1 kapcsolat van.

```

SELECT anyag.azonosító, MAX(anyag.neve) AS anyagnév,
      SUM(rendelés.darab * szerkezet.mennyiség)
      AS [összes anyag],
      MIN(anyag.mért_egys) AS mértékegység
FROM rendelés, szerkezet, anyag WHERE rendelés.kód =
szerkezet.kód
      AND szerkezet.azonosító = anyag.azonosító
      AND rendelés.dátum = #11/10/2001#
GROUP BY anyag.azonosító;

```

Amint a fenti megoldásból láthatjuk, az SQL nem követeli meg a táblák összekapcsolásánál az 1 : N kapcsolat meglétét. A WILRF záradékban bármilyen feltétel lehet. A feltétel a táblák direktszorzatából választja ki azokat a sorokat, amelyek a feltételt kielégítik. Ha a relációjel két oldalán két különböző táblából származó mezőnév van, akkor a két táblából olyan sorok kerülnek be az eredmény- táblába, amelyek a feltételnek megfelelnek, és ez független attól, hogy van-e és, ha van, milyen kapcsolat \an a két tábla között.

A kulcs - külső kulcs kapcsolatnál általánosabb fogalom a két tábla metszet tulajdonságainak a halmaza, amelyik a két táblából származó azon mezőpárokat tartalmazza, amelyeknek közös a tulajdonsághalmazuk. A kulcs - külső kulcs pár **nem** van a metszet tulajdonságok halmazában, de nem minden metszet tulaj- . iság kulcs - külső kulcs pár.

akorló feladatok

1. Határozzuk meg, hogy a legnagyobb rendelési számú megrendelésnek a teljesítéséhez rendelkezésre áll-e minden anyagból a megfelelő készlet!
2. Határozzuk meg, hogy az el nem készült termékeknek mennyi az anyagszükséglete!
3. Határozzuk meg, hogy egy adott rendelésszámú megrendelésnek mennyi az anyagszükséglete. A rendelési szám legyen paraméter.

16. Legyártható-e egy termék?

1. Határozzuk meg, hogy az „AA1” kódú termék legyártható-e, azaz a termék előállításához szükséges anyagmennyiség rendelkezésre áll-e!

Először határozzuk meg, hogy az „AA1” kódú termék előállításához milyen anyagokra van szükség. A termék nevét és a szükséges anyagok azonosítóját és nevét is ki kell keresni.

```
SELECT név, termék.kód, neve anyag.azonosító
FROM termék, szerkezet, anyag
WHERE termék.kód = szerkezet.kód
      AND szerkezet.azonosító = anyag.azonosító AND
      termék.kód = 'AAT';
```

Az „AA1 ” kódú termék akkor gyártható le, ha minden a termék előállításához szükséges anyagból legalább akkora a készlet, mint amennyi a termék szerkezetét leíró mennyiség mező értéke.

```
SELECT név, termék.kód, neve, anyag.azonosító, mennyiség,
      készlet, Hőmennyiség > készlet,
      'nem gyártható le', 'legyártható') AS gyarthat
FROM termék, szerkezet, anyag WHERE termék.kód =
      szerkezet.kód
      AND szerkezet.azonosító = anyag.azonosító
      AND termék.kód = 'AAT';
```

Az eredmény táblának annyi sora lesz, mint ahányféle anyagból áll az „AA1” kódú termék, és akkor gyártható le az „AA1” kódú termék, ha minden anyagból megvan a szükséges mennyiség, azaz egyszer sem jelenik meg a „nem gyártható le” szöveg. Az eredmény táblában megvan a szükséges információ, de ez így nagyon nehezen kezelhető.

2. Határozzuk meg, hogy egy termék legyártható-e, és ha igen, hány darab gyártható le! A termék kódját paraméterként adjuk meg a lekérdezés idején!

Ha nem egy konkrét termék legyárthatóságát szeretnénk megvizsgálni, akkor a termék kód helyett egy paramétert használhatunk, amelynek az értékét a lekérdezés kiértékelése idején kell megadni (p_kód).

```

SELECT név, termék.kód, neve, anyag.azonosító, mennyiség,
       készlet, Hőmennyiség > készlet,
       'nem gyártható le', 'legyártható') AS gyárthat,
       Int(készlet/mennyiség) AS darab
FROM termék, szerkezet, anyag
WHERE termék.kód = szerkezet.kód
      AND szerkezet.azonosító = anyag.azonosító AND
      termék.kód = p_kód;

```

Ez így áttekinthetetlen, bár minden információt tartalmaz. A termékhez szükséges minden anyagnál azt mondja meg az eredménytábla, hogy az adott anyag miatt mennyi az a termékmennyiség, amennyi legyártható. Az adott termékből annyi gyártható le, amennyi a darab legkisebb értéke, és ha ez az érték nulla, akkor a termék nem gyártható le.

```

SELECT          MAX(név),          MAX(termék.kód),
       If(Int(MIN(készlet/mennyiség)) = 0,
       'nem gyártható le', 'legyártható') AS gyárthat,
       Format(Int(MIN(készlet/mennyiség)), "0") &
       ' darab gyártható le' AS darab
FROM termék, szerkezet, anyag
WHERE termék.kód = szerkezet.kód
      AND szerkezet.azonosító = anyag.azonosító AND
      termék.kód = p_kód;

```

A Formát függvény átalakít egy számot karaktersorozattá. A „0” formátumkód megadja, hogy legalább egy számjegyből álljon a karaktersorozat, vezető nullákat ne tartalmazzon. Az Int függvény a szám egészrészét adja vissza.

A név és termék.kód mezők maximumát tudjuk csak kiírni, mert az eredménytábla összesen egy sorból áll, de ez nem zavaró, mert, mint a WHFRE záradék utolsó feltételéből látható, most minden név, illetve termék.kód mező egyenlő egymással.

3. Minden termékre határozzuk meg, hogy legyártható-e vagy sem!

Figyelem! Ha egy terméket legyártunk, megváltoznak a készletek, és már nem ugyanazok a termékek lesznek legyárthatóak. A feladatban csak azt vizsgáljuk, hogy egy adott helyzetben, melyek azok a termékek, amelyek közül bármelyik egyártható. Ha egy terméket le akarunk gyártani, a felhasználandó anyag mennyiségével csökkenteni kell a készletet, amikor az anyagot kivételezzük a raktárból, és a lekérdezés újbóli lefuttatásával megkaphatjuk, hogy az új helyzetben melyek a csökkentett anyagkészletből legyártható termékek.

I. Megoldás

Először készítsünk egy lekérdezést a SZERKEZET és ANYAG táblából, amely eredmény táblája csak annyiban különbözik a SZERKEZET táblától, hogy van még egy oszlopa, amely minden termékre megmutatja, hogy az adott termékhez az adott anyagból rendelkezésre áll-e a szükséges mennyiség. A mező értéke legyen 1, ha a sor által meghatározott termékhez az adott anyagból elegendő mennyiségben áll rendelkezésre, különben legyen 0.

```
SELECT kód, szerkezet.azonosító, mennyiség, If(mennyiség >
      készlet, 0, 1) AS gyárthat
FROM szerkezet, anyag
WHERE szerkezet.azonosító = anyag.azonosító;
-> SZERKEZET_GYÁRT{kód, azonosító, mennyiség, gyárthat}
```

Ha a SZERKEZET GYÁRT táblában egy termékkódhoz tartozó sorok száma egyenlő ugyanezen sorokban levő gyárthat mezők összegével, azaz nincs közöttük 0 értékű mező, akkor a termék legyártható.

```
SELECT név, termék.kód,
      If(SUM(gyárthat) = COUNTf,
        'legyártható', 'nem gyártható le') AS legyártható
FROM termék, szerkezet gyárt
WHERE termék.kód = szerkezet_gyárt.kód
GROUP BY termék.kód, név;
```

II. Megoldás

Ha csak azokat a termékeket akarjuk kigyűjteni, amelyek nem gyárthatóak le, és nem érdekel, hogy miért nem, akkor elég megnézni, hogy van-e olyan sora a szerkezet táblának, amely akadályozza a gyártást. Természetesen lehet olyan termék, amelyet többféle anyag hiánya miatt sem tudunk legyártani, ezért kell a lekérdezésben a DISTINCT szó is.

```
SELECT DISTINCT termék.kód, név,
      'nem gyártható le' AS gyárthat
FROM termék, szerkezet, anyag
WHERE termék.kód = szerkezet.kód
      AND szerkezet.azonosító = anyag.azonosító AND
      mennyiség > készlet;
      NEM_GYÁRTHATÓ{kód, név, gyárthat}
```

Amelyik termék legyártható, az nincs benne a NEM GYÁRTHATÓ eredménytáblájában.

```

SELECT termék.kód, név, 'legyártható' AS gyárthat
FROM termék
WHERE kód NÖT IN (SELECT kód
                  FROM nem_gyártható);

```

A két lekérdezést összeépíthetjük egy lekérdezésbe, és megkapjuk, hogy éppen most melyik tennék gyártható le.

```

SELECT termék.kód, név, 'legyártható' AS gyárthat
FROM termék
WHERE kód NÖT IN
  (SELECT kód
   FROM szerkezet, anyag
   WHERE szerkezet.azonosító = anyag.azonosító AND
        mennyiség > készlet);

```

A belső SELECT utasítás most egyszerűbb, mint a NEM GYÁRTHATÓ lekérdezésben, mert itt csak a termék kódját akartuk meghatározni.

Vegyük észre, hogy az I. megoldás és II. megoldás nem ad feltétlenül azonos eredményt. Ali. megoldásba, ellentétben I. megoldással, belekerülnek azok a termékek is, mint legyárthatók, amelyeknek nincs szerkezeti leírása. A feladatunk értelmében azonban ilyen termék nem lehet.

III. Megoldás

Úgy is fogalmazhatjuk ezt a kérdést, hogy akkor gyártható le egy termék, ha a termékhez tartozó szerkezet és a hozzá tartozó anyagsorban mindegyikre igaz, hogy a készletből kivonva mennyiséget az eredmény nagyobb vagy egyenlő nullánál.

```

SELECT termék.kód, név, 'legyártható' AS gyárthat
FROM termék
WHERE 0 <= ALL
  (SELECT készlet - mennyiség FROM szerkezet, anyag
   WHERE szerkezet.azonosító = anyag.azonosító
        AND szerkezet.kód = termék.kód);

```

Megjegyzés

Az ACCESS 97 a fenti lekérdezés WHERE feltételében nem engedi meg a 0 jsználatát, helyette +0 kifejezést kell használni. Az ALL és az ANY bal oldalán -.egenged bármilyen mezőnevet, konstans vagy kifejezést, kivéve 0-t, illetve -0-t. Az ACCESS 2002 már nem tartalmaz ilyen kikötést.

Akkor nem gyártható le a termék, ha van a fenti módon kiválasztott sorok között akár egyetlen olyan sor, amire nem teljesül az ott megadott feltétel, azaz van közöttük olyan sor, ahol a készlet - mennyiség kisebb nullánál.

```
SELECT termék.kód, név, 'nem gyártható le' AS gyárthat
FROM termék
WHERE 0 > ANY
      (SELECT készlet - mennyiség FROM szerkezet, anyag
       WHERE szerkezet.azonosító = anyag.azonosító
        AND szerkezet.kód = termék.kód);
```

Az előző két lekérdezés egyesítésével, és például név szerinti rendezésével, megkaphatjuk a termékek teljes listáját, hiszen minden termék szerepel és csak egyszer valamelyik lekérdezésben, ugyanis egy termék vagy legyártható vagy nem.

```
SELECT termék.kód, név, 'legyártható' AS gyárthat
FROM termék
WHERE 0 <= ALL
      (SELECT készlet - mennyiség FROM szerkezet, anyag
       WHERE szerkezet.azonosító = anyag.azonosító AND
        szerkezet.kód = termék.kód)

UNIÓN
SELECT termék.kód, név, 'nem gyártható le' AS gyárthat
FROM termék
WHERE 0 > ANY
      (SELECT készlet - mennyiség FROM szerkezet, anyag
       WHERE szerkezet.azonosító = anyag.azonosító AND
        szerkezet.kód = termék.kód)

ORDER BY név;
```

Az UNIÓN utasítással egyesített két lekérdezésnek kompatibilisnek kell lenni, ami azt jelenti, hogy ugyanannyi oszlopának kell lenni, és az egymás alá kerülő mezők típusának és hosszának is meg kell egyezni.

Az eredménytábla a mezőneveket az első SELECT utasításból örökli.

A rendezés az egész eredménytáblára vonatkozik, (nemcsak az előtte lévő SELECT utasításban kiválasztott sorokra, hanem az egyesített eredményre).

Az UNIÓN utasítás az összekapcsolt két SELECT utasítás eredménytáblájának a soraiból halmazelméleti értelemben készít egy egyesített eredménytáblát, az azonos sorokból csak egyet hagy meg, és a sorok sorrendje elvileg meghatározatlan. Az ORDER BY záradék az így elkészített eredménytáblára vonatkozik.

IV. Megoldás

Természetesen ugyanezt az eredményt megkaphatjuk a következő módon is, ha felhasználjuk a II. megoldásban előállított eredménytáblát:

```
SELECT termék.kód, név, 'nem gyártható le' AS gyártható FROM
    termék
    WHERE kód IN (SELECT kód
                  FROM nem_gyártható)
UNIÓN
SELECT termék.kód, név, 'legyártható' AS gyártható
    FROM termék
    WHERE kód NOT IN (SELECT kód
                     FROM nem_gyártható) ORDER
    BY név;
```

V. Megoldás

A IV. megoldásban alkalmazott két SELECT utasítás és UNIÓN helyett egyetlen SELECT utasítással is megfogalmazható a feladat. Ebben az esetben a SELECT mezőlistájában lévő gyárthat mező értékét egy feltétel alapján kell meghatározni, amiben belül van még egy beágyazott SELECT utasítás (ez is két SELECT).

```
SELECT termék.kód, név,
    If(kód IN (SELECT kód
              FROM nem_gyártható),
      'nem gyártható le', 'legyártható') AS gyártható
    FROM termék
    ORDER BY név;
```

VI. Megoldás

Máshogy megfogalmazva, nem gyártható le a termék, ha létezik legalább egy, a gyártást akadályozó anyag.

```
SELECT termék.kód, név,
    If(EXISTS (SELECT kód
              FROM szerkezet, anyag
              WHERE szerkezet.azonosító = anyag.azonosító AND
                    mennyiség > készlet AND szerkezet.kód =
                    termék.kód),
      'nem gyártható le', 'legyártható') AS gyárthat
    FROM termék
    ORDER BY név;
```

Vegyük észre, hogy itt a belső SELECT utasítást kiegészítettük egy - a külső SELECT utasításból származó termék.kód mezőt tartalmazó - feltétellel. Ennek következtében a SELECT utasítás mezőlistájában szereplő belső SELECT utasítás nem értékelhető ki a külső SELECT utasítás végrehajtása előtt, minden termékre, a sor megjelenítése előtt újra végrehajtódik. A megoldás szép, de feleslegesen bonyolult.

Gyakorló feladatok

1. Határozzuk meg, hogy van-e elegendő készlet 2 darab „AA1” és 3 darab „AA2” kódú termék legyártásához!
2. Határozzuk meg, hogy az „axo” partnerkódú megrendelő utolsó megrendelése teljesíthető-e, azaz a megrendelésének minden terméke a kívánt darabszámban legyártható-e!
3. Határozzuk meg, hogy az „axo” partnerkódú megrendelő utolsó megrendelése teljesíthető-e, figyelembe véve, hogy lehetnek olyan termékek is, amelyek már elkészültek!

17. Adott időintervallumban megrendelt, illetve meg nem rendelt termékek

1. Határozzuk meg, hogy melyik terméket (kód, elnevezés) rendelt meg már valaki (legalább egyszer megrendelték)!

I. Megoldás

A TERMÉK és RENDELÉS tábla összetartozó soraiból a különböző sorokat kiírjuk (kell a DISTINCT, mert egy terméket többször is megrendelhettek, az egyesített táblának sem a termék kódja, sem a neve nem kulcsa, hanem a rend_szám és a kód együttesen a kulcs):

```
SELECT DISTINCT termék.kód, név
FROM termék, rendelés WHERE termék.kód = rendelés.kód;
```

II. Megoldás

Nem szükséges a direkt szorzat, mert csak a TERMÉK táblából szeretnénk listázni adatokat (a belső SELECT csak egyszer értékelődik ki):

```
SELECT kód, név
FROM termék
WHERE kód IN (SELECT kód
              FROM rendelés);
```

.7. Megoldás

A belső SELECT-ben minden termékre kiszámítjuk, hogy az adott termékre '■'iny rendelés volt, és, ha ez az érték nagyobb, mint nulla, akkor volt rá megrendelés (a belső SELECT a TERMÉK tábla minden sorára újra kiértékelődik):

```
SELECT kód, név
FROM termék
WHERE 0 < (SELECT COUNT(*)
           FROM rendelés
           WHERE termék.kód = rendelés.kód);
```

T Megoldás

\ belső SELECT-ben elég csak azt vizsgálni, hogy a RENDELÉS táblában van-e gfelelő sor (a belső SELECT itt is kiértékelődik a TERMÉK tábla minden sorára):

```
SELECT kód, név
      FROM termék
      WHERE EXISTS (SELECT *
                    FROM rendelés
                    WHERE termék.kód = rendelés.kód);
```

2. Határozzuk meg, hogy melyik termékre (kód, elnevezés) nem történt megrendelés még soha (senki sem rendelte még meg)!

I. Megoldás

Azoknak a termékeknek a listája, amelyek nem szerepelnek a RENDELÉS táblában:

```
SELECT kód, név
      FROM termék
      WHERE kód NOT IN (SELECT kód
                       FROM rendelés);
```

II. Megoldás

Azoknak a termékeknek a listája, amelyeket 0-szor fordulnak elő a RENDELÉS táblában (a belső SELECT a TERMÉK tábla minden sorára kiértékelődik):

```
SELECT kód, név
      FROM termék
      WHERE 0 = (SELECT COUNT(*)
                FROM rendelés
                WHERE termék.kód = rendelés.kód);
```

III. Megoldás

A belső SELECT-ben csak azt vizsgáljuk, hogy nem létezhet olyan sor a RENDELÉS táblában, amely az adott termékre vonatkozik (a belső SELECT itt is kiértékelődik a TERMÉK tábla minden sorára):

```
SELECT kód, név
      FROM termék
      WHERE NOT EXISTS (SELECT *
                       FROM rendelés
                       WHERE termék.kód = rendelés.kód);
```

Megjegyzés

Fontos megjegyezni a 1. és 2. feladat közötti különbséget:

- az 1. feladatban azt kerestük, hogy a RENDELÉS táblában benne van-e egy termékkód,
- a 2. feladatban pedig azt, hogy a RENDELÉS táblából hiányzik-e egy termékkód.

Az 1. feladat I. megoldásában, a TERMÉK és RENDELÉS táblák „értelmes” (egyenlőségen alapuló természetes) összekapcsolásában csak azok a termékkódok maradnak benne, amelyek mind a TERMÉK, mind a RENDELÉS táblában előfordultak. Arra már nem adnak semmilyen információt, hogy melyek voltak azok a termékkódok, amelyek az összekapcsolás miatt kiestek az eredménytáblából. Azt tehát, hogy mit nem rendeltek meg, nem tudjuk az összekapcsolt táblából megállapítani, ezért az 1. feladat I. megoldásához hasonlóval hiába próbálkoznánk a 2. feladtnál.

Ha az a feladat, hogy azt keressük meg, hogy melyik termékkód nem szerepel a RENDELÉS táblában, azt csak a TERMÉK táblából kiindulva, a termékkódok egyenként történő megvizsgálásával tudjuk eldönteni.

3. Határozzuk meg, hogy melyik termék (kód, elnevezés) nem készült el legalább egy éve! (Azok a termékek tesznek eleget a feltételnek, amelyeknek az elkészítését egyszer sem vállalták olyan határidőre, amely az egy évvel ezelőtti dátumnál nagyobb.)

I. Megoldás

Minden termék legutolsó elkészítése (az a termék, amit nem rendeltek meg soha, nem lesz benne ebben a listában):

```
SELECT kód, MAX(dátum) AS utolsó  
FROM rendelés  
GROUP BY kód;
```

Amint az utolsó éven belülre vállaltak:

```
SELECT kód, MAX(dátum) AS utolsó  
FROM rendelés GROUP BY kód HAVING MAX(dátum) >  
DateAdd("yyyy",-1,Date());  
EGY_ÉVEN_BELÜL{kód, utolsó}
```

Ami nincs benne az egy éven belüliekben (ebben már benne lesznek azok a termékek is, amelyeket sohasem készítettek el):

```
SELECT kód, név
      FROM termék
      WHERE kód NŐT IN (SELECT kód
                        FROM egy_éven_belül);
```

II. Megoldás

Ha az utolsó elkészítésre igaz, hogy egy éven belül volt, akkor az előző megoldás első két SELECT utasítása helyett elég kigyűjteni az egy éven belüli elkészítéseket, ebben benne lesznek az utolsó elkészítések is (egy kód, különböző dátumokkal, többször is előfordulhat a listában):

```
SELECT kód, dátum
      FROM rendelés
      WHERE dátum > DateAdd("yyyy",-1,Date());
                + EGY_ÉVEN_BELÜL_2{kód, utolsó}
```

Ami nincs benne az egy éven belüliekben:

```
SELECT kód, név
      FROM termék
      WHERE kód NŐT IN (SELECT kód
                        FROM egy_éven_belül_2);
```

III. Megoldás

Fgy SELECT utasításba összevonva a II. megoldást:

```
SELECT kód, név
      FROM termék
      WHERE kód NŐT IN (SELECT kód
                        FROM rendelés
                        WHERE dátum > DateAdd("yyyy",-1 ,Date()));
```

IV. Megoldás

A TERMFK táblában található minden termékre megvizsgáljuk, hogy van-e olyan elkészítés, ami egy éven belül történt (ha nincs, akkor ez a termék nem készítettük el az elmúlt évben):

```
SELECT kód, név
      FROM termék
      WHERE NOT EXISTS (SELECT *
                        FROM rendelés
                        WHERE termék.kód = rendelés.kód
                        AND dátum > DateAdd("yyyy",-1,Date()));
```

Megjegyzés:

Ezeknél a feladatoknál két kérdést kell mindig megvizsgálni:

1. A lekérdezés eredménye milyen termékcsoportra vonatkozik?
 - Ha a teljes termékcsoportból kell kiválasztani a termékeket, akkor a TERMÉK táblából kell kiindulni.
 - Ha a megrendelt termékek közül kell valamennyit kiíratni, akkor RFNDELÉS táblából. Annak eldöntéséhez, hogy milyen terméket nem rendeltek meg, csak a TERMÉK táblából kiindulva kaphatunk információt.

2. Milyen időintervallumra vonatkozik a kérdés, illetve a feltétel?

Például: ha a kérdésben két időpont szerepel (ma, egy évvel ezelőtt), akkor ez három időintervallumot határoz meg (egy évvel ezelőttig, egy évvel ezelőttől a mai napig, a mai naptól).

Minden időintervallumra vonatkozhat valamilyen feltétel, és ezeket együttesen kell kielégíteni.

A jelen feladatban a teljes termékcsoportból kellett kiindulni, és a feltétel egy időintervallumra vonatkozott, egy évvel ezelőttől akármeddig. A feladat azzal, hogy több mint egy évvel ezelőtt mi történt, nem foglalkozik.

A három feladat összehasonlítása:

	időintervallum		a termékkód	
	egy éve	ma	TERMÉK-ben	RENDELÉS-ben
	i	i		
1. feladat	----- 1 -----	1 ----- ►	benne van	benne van
2. feladat	----- 1 -----	1 ----- ►	benne van	nincs benne
3. feladat	1-----	1----- ►	benne van	nincs benne

Gyakorló feladatok

1. Készítsünk egy paraméteres lekérdezést, amelyik megmutatja, hogy 2001. évben egy partner milyen termékeket rendelt meg!
2. Készítsünk egy paraméteres lekérdezést, amelyik megmutatja, hogy 2001. évben egy partner milyen termékeket nem rendelt meg!
3. Készítsünk egy paraméteres lekérdezést, amelyik megmutatja, hogy melyek azok a termékek, amelyeket 2001. évben egy partner megrendelt, de 2002. évben nem rendelt meg!

18. Egy adott termékcsoporthoz megrendelt megrendelések listája

1. Keressük meg azokat a megrendeléseket, amelyekben megrendelték az összes olyan terméket, amelyeknek a kódja egy KERES nevű táblában van! (elég a megrendelés számát kiírni)

Először készítsük el a KERES táblát, amely tetszésszerűen, de a TERMÉK táblában benne lévő, termékkódot tartalmaz.

KERES{kód}

I. Megoldás

Ez a feladat a 9. feladat általánosítása. Az általánosított feladat megoldásánál nem tudjuk előre, hogy hány termékkód van a KERES táblában, így olyan megoldást kell választani, ahol RENDELÉS táblát csak egyszer használtuk fel. Induljunk ki a 9. feladat I. megoldásából, azonban ahhoz képest most két különbség van:

- a termékkódok egy külön táblában vannak, ezért egy beágyazott SELECT utasítással gyűjthetők ki,
- a csoportokból nem azokra van szükségünk, amelyeknek két soruk van, hanem azokra, amelyeknek annyi soruk van, mint ahány sora van a KERES táblának, tehát a HAVING záradékban is egy beágyazott SELECT utasításra van szükségünk.

```
SELECT rend_száma
FROM rendelés
WHERE kód IN (SELECT kód
              FROM keres)
GROUP BY rend_száma
HAVING Count(kód) = (SELECT Count(*)
                    FROM keres);
```

II. Megoldás

A RENDELÉSFEJ és a KERES tábla direktszorzatából kaphatunk egy olyan listát, amelyik minden rendelésszámmal hozzárendeli az összes keresett termékkódot.

```
SELECT rend_száma, kód
FROM rendelésfej, keres;
```

Ha ebből egy konkrét rendelésszámhoz tartozó minden sor benne van a RENDELÉS táblában, akkor a rendelésben benne vannak a keresett termékkódok is. Átfogalmazva, ha ebből a táblából akár csak egyetlen sor is nem szerepel a megrendelések között, akkor az a megrendelés nem teljesíti a feladatban megfogalmazott feltételt. Rendelésenként a meg nem rendelt termékek:

```
SELECT rend_szám, kód
FROM rendelésfej, keres
WHERE rend_szám & kód NŐT IN
      (SELECT rend_szám & kód
       FROM rendelés);
```

Vegyük észre, hogy a belső SELECT utasításban nem egy mező értékeiből képeztünk egy halmazt, hanem egy kifejezés értékeiből, a rendelésszám és a termékkód karaktersorozat egyesítéséből. Erre azért volt szükségünk, mert egy adott sorra egyszerre kellett teljesülni annak, hogy a rendelésszám és a termékkód értékpár együttesen nem szerepel a belső SELECT utasítással előállított belső tábla egyetlen sorában sem.

A mezőnevek minősítésére nem volt szükség, a hivatkozások egyértelműek, mert a belső SELECT utasításban mind a rend szám, mind a kód a RENDELÉS táblához tartozik, a külső SELECT utasításban a rend szám a RENDELÉSFEJ táblához, a kód a KERES táblához tartozik.

Azok a megrendelések tartalmazznak minden keresett terméket, amelyek nincsenek benne ennek a táblának a rend_szám oszlopában.

```
SELECT rend_szám
FROM rendelésfej
WHERE rend_szám NŐT IN
      (SELECT rend_szám
       FROM rendelésfej, keres
       WHERE rend_szám & kód NŐT IN
            (SELECT rend_szám & kód
             FROM rendelés));
```

III. Megoldás

Elegánsabb, de kevésbé hatékony megoldást kapunk, ha használjuk az EXISTS predikátumot, és szó szerint leírjuk SQL nyelven a következő mondatot:

Vizsgáljuk meg a RENDELÉS tábla egy rendelésszámát. A keresett feltételnek akkor felel meg a rendelésszám, ha üres az a lista, amelyet úgy kapunk, hogy a vizsgált rendelésszám és keresett termékkódpárok mindegyikét megvizsgáljuk, hogy van-e közöttük olyan, amelyik nem szerepel a vizsgált rendelésszámhoz tartozó rendelésben. Ha valamelyik keresett kód nem szerepel a vizsgált rende

lésben, akkor a termék kódja bekerül a kódlistába, így a kódlista nem lesz üres, a vizsgált rendelésszám tehát nem felel meg a feltételnek.

```
SELECT rend_szám
FROM rendelésfej vizsgált
WHERE NŐT EXISTS (SELECT kód
                  FROM rendelésfej, keres
                  WHERE rend_szám = vizsgált.rend_szám AND kód NŐT IN
                  (SELECT kód
                   FROM rendelés
                   WHERE rend_szám = vizsgált.rend_szám));
```

Inkább szép (annak, aki szereti az ilyen összetett gondolatok precíz megfogalmazását), mint hasznos az ilyen bonyolultságú megoldás.

%

Gyakorló feladatok

1. Határozzuk meg, hogy melyik partner rendelt meg minden terméket!
2. Határozzuk meg, hogy melyik termékben van benne minden olyan anyag, amelyik benne van az „AAI” kódú termékben!
3. Határozzuk meg, hogy melyik termékben van benne minden olyan anyag legalább akkora mennyiségben, mint amelyik benne van az „AAI” kódú termékben!

19. Az ALL, ANY és EXISTS predikátum összehasonlítása

1. Keressük ki azokat a megrendelőket, akinek minden megrendelése 10000.- Ft felett volt!

Kiindulásként gyűjtsük ki az összes megrendelőt, akinek volt valamilyen értékű megrendelése. Fontos, hogy a kiindulásnál különbséget tegyünk egy adatbázisban létező megrendelő törzset tartalmazó táblázatban lévő megrendelők (mert ezeknek esetleg egyetlen megrendelése sem volt) és azok között, akiknek volt megrendelése. Akinek van nyilvántartott rendelése, annak a kódja a RENDELÉSFEJ táblában van. Ha nem gondolunk erre a tényre, és egy PARTNER táblából indulnánk ki, a feldolgozásba bekerülhet olyan megrendelő, akinek nincs megrendelése (már archiváltuk a megrendelését, vagy valamilyen más célból, mint potenciális megrendelőt tartjuk nyilván). A potenciális megrendelőként nyilvántartottakra igaz lehet, hogy nincs 10000.- Ft alatti megrendelése, hiszen nincs egyetlen megrendelése sem. A mi adatbázisunkban, mint a könyv elején tisztáztuk, lehetne PARTNER tábla is, de akkor is a RENDELÉSFEJ táblából kell kiindulni.

```
SELECT DISTINCT partner_kód  
FROM rendelésfej;
```

PARTNER{partner_kód}

I. Megoldás

Végig kell menni minden partner_kód-on, és az adott partner kód akkor jó, ha a hozzátartozó összes megrendelés megrendelésenként összesített értéke nagyobb, mint 10000.- Ft. Egy adott megrendelés összesített értékét egy belső SELECT utasítással számítsuk ki.

```
SELECT partner_kód  
FROM partner  
WHERE 10000 < ALL  
      (SELECT SUM(darab*ár)  
       FROM rendelésfej, rendelés, termék  
       WHERE rendelésfej.partner_kód = partner.partner_kód  
       AND rendelésfej.rend_száma =  
                                     rendelés.rend_száma  
       AND rendelés.kód = termék.kód  
       GROUP BY rendelésfej.rend_száma);
```

A belső SELECT utasítás a külső SELECT utasítás által kiválasztott egyetlen partner megrendeléseinek rendelésszámonként összesített értékét adja vissza. A belső SELECT által visszaadott összes számra igaznak kell lennie, hogy mindegyik nagyobb, mint 10000.

II. Megoldás

A megoldáshoz eljuthatunk a feltételnek tagadó formában való megfogalmazásával is, azaz azokat a megrendelőket keressük, akiknek nem létezik olyan megrendelésük, amelyiknek a megrendelésenként összesített értéke kisebb vagy egyenlő, mint 10000.- Ft.

```
SELECT partner_kód
FROM partner WHERE NÖT EXISTS
(SELECT SUM(darab*ár)
FROM rendelésfej, rendelés, termék
WHERE rendelésfej. partner_kód = partner.
partner_kód AND rendelésfej.rend_szám =
rendelés.rend_szám
AND rendelés.kód = termék.kód
GROUP BY rendelésfej.rend_szám
HAVING SUM(darab*ár) <= 10000);
```

III. Megoldás

Igaz az, hogy, ha a partner minden megrendelése nagyobb volt, mint 10000.- Ft, akkor a legkisebb is nagyobb volt 10000.- Ft-nál.

Számoljuk ki először minden partnernek rendelésszámonként a megrendelését, utána válasszuk ki azokat a partnereket, akinek a legkisebb megrendelése is nagyobb, mint 10000.- Ft.

```
SELECT rendelésfej.rend_szám, SUM(darab*ár) AS rend_ár,
MAX(partner_kód) AS kód
FROM rendelésfej, rendelés, termék
WHERE rendelésfej.rend_szám = rendelés. rend_szám
AND rendelés.kód = termék.kód
GROUP BY 'endelésfej.rend_szám;
REND_ÁR{rend_szám, rend_ár, kód}
```

A RENDELÉSFEJ táblában egy rend_szám-hoz egyetlen partner kód tartozik. A RENDELÉSFEJ, RENDELÉS és TERMÉK tábla direktszorzatában ugyanaz a rend_szám és partner kód annyiszor fog szerepelni, ahány terméket rendeltek

meg az adott rend szám-on. Ha tehát rend szám-ra csoportosítunk, egy rendszám-on belül mindig ugyanaz a partner kód szerepel, ezért írhatjuk a SELECT utasítás mezőlistájába a MAX(partner_kód) vagy a MIN(partner_kód) kifejezést, mindkettő ugyanazt eredményezi.

Ebből a REND_ÁR táblából kell kiválogatni azokat a partnereket, akiknek a legkisebb rendelési ára is nagyobb, mint 10000.- Ft.

```
SELECT kód, MIN(rend_ár)
FROM rend_ár
GROUP BY kód HAVING MIN(rend_ár) > 10000;
```

2. Keressük ki azokat a megrendelőket, akiknek van olyan megrendelésük, amelyik 10000.- Ft vagy az alatt volt!

Ez a feladat az előző tagadása, azaz nem igaz, hogy a megrendelőnek minden megrendelése 10000.- Ft felett volt, előfordult olyan is, hogy alatta volt. Lehet, hogy mindegyik alatta volt. A feladat szövege semmit sem mond a 10000.- Ft feletti megrendelésekről.

I. Megoldás

Azt kell megvizsgálni, hogy létezik-e olyan megrendelése az adott partnernek, amelyik 10000.- Ft vagy az alatti.

```
SELECT partner_kód
FROM partner
WHERE EXISTS
  (SELECT SUM(darab*ár)
   FROM rendelésfej, rendelés, termék
   WHERE rendelésfej.partner_kód = partner.partner_kód
     AND rendelésfej.rend_szám =
                                     rendelés.rendszám
     AND rendelés.kód = termék.kód
   GROUP BY rendelésfej.rend_szám HAVING
    SUM(darab*ár) <= 10000);
```

II. Megoldás

A feladatot úgy is fogalmazhatjuk, hogy az adott partner megrendeléseit kell kiválasztani, majd megnézni, hogy volt-e közöttük 10000.- Ft-nál kisebb vagy egyenlő értékű megrendelés.

```
SELECT partner_kód FROM partner WHERE 10000 >= ANY
      (SELECT SUM(darab*ár)
       FROM rendelésfej, rendelés, termék
       WHERE rendelésfej. partner_kód =
                                partner.    partner_kód
       AND rendelésfej.rend szám =
                                rendelés.rend_szám
       AND rendelés.kód = termék.kód
       GROUP BY rendelésfej.rend_szám);
```

III. Megoldás

Ha a partnernek volt olyan megrendelése, amelyik kisebb vagy egyenlő 10000.- Ft-nál, akkor a legkisebb megrendelésének az értéke is kisebb vagy egyenlő volt, mint 10000.- Ft.

Használjuk fel az első feladatnál előállított REND_ÁR eredménytáblát, amelyik tartalmazza a partnerek megrendeléseinek megrendelésenkénti összesített árát.

```
SELECT kód, MIN(rend_ár)
FROM rend_ár GROUP BY kód
HAVING MIN(rend_ár) <= 10000;
```

3. Keressük ki azokat a megrendelőket, akiknek minden megrendelése kisebb vagy egyenlő, mint 10000.- Ft!

I. Megoldás

```
SELECT partner_kód
FROM partner
WHERE 10000 >= ALL
      (SELECT SUM(darab*ár)
       FROM  rendelésfej, rendelés, termék WHERE
       rendelésfej. partner_kód = partner.partner_kód
       AND rendelésfej.rend_szám =
                                rendelés.rend_szám
       AND rendelés.kód = termék.kód
       GROUP BY rendelésfej.rend_szám);
```

II. Megoldás

Vigyázzunk, az ANY vagy ALL predikátumot tartalmazó logikai kifejezés tagadása sajnos csak a relációidét tagadja és nem az egész logikai kifejezést!

```
SELECT partner_kód
FROM partner
WHERE NŐT 10000 < ALL
      (SELECT SUM(darab*ár)
       FROM rendelésfej, rendelés, termék
       WHERE rendelésfej. partner_kód =
                                partner. partner_kód
         AND rendelésfej.rend_szám =
                                rendelés. rend_szám
         AND rendelés.kód = termék.kód
       GROUP BY rendelésfej.rend_szám);
```

Figyelem! Ez a lekérdezés nem az első feladat tagadása, hanem ugyanaz, mint az előző lekérdezés.

III. Megoldás

```
SELECT kód, MAX(rend_ár)
FROM rend_ár
GROUP BY kód
HAVING MAX(rend_ár) <= 10000;
```

Összefoglalva:

A partner megrendelése:

	10000.-Ft-nál nagyobb	10000.- Ft-nál kisebb vagy egyenlő
1. feladat	mind	nincs
2. feladat	lehet	van
3. feladat	nincs	mind

Xz 1. feladat tagadása a 2. feladat, és nem a 3. feladat. A 3. feladat tagadása az lenne, hogy a megrendelőnek van 10000.- Ft-nál nagyobb értékű megrendelése. Ez lenne a 4. feladat, de ennek a megoldását már az olvasóra bizzuk. A 4. feladatban benne lennének azok is, akiknek minden megrendelése 10000.- Ft feletti, de benne vannak azok is, akiknek van 10000.- Ft alatti megrendelése is.

Táblázatba foglalva, hogy az egyes lekérdezésekbe a megrendelések összértéke szerint milyen megrendelők kerültek bele:

10000.- Ft-nál nagyobb	10000.- Ft-nál kisebb vagy egyenlő	1. feladat	2. feladat	3. feladat	4. feladat
van	van		benne van		benne van
van	nincs	benne van			
nincs	van		benne van	benne van	
nincs	nincs	ezt az esetet a lekérdezésekből kizártuk			

Kapcsolatok a feladatok között:

- az 1. feladat tagadása a 2. feladat,
- a 3. feladat tagadása a 4. feladat.

Gyakorló feladatok

1. Keressük meg azokat a termékeket, amelyeket felépítő minden anyagának a kódja „A” karakterrel kezdődik!
2. Keressük meg azokat a termékeket, amelyben van olyan anyag, amelyből a beépített mennyiség összértéke 1000.- Ft felett van!
3. Keressük meg azokat a termékeket, amelyből sohasem vettek egyszerre többet, mint egy darabot!

20. Csoportokba sorolás külső feltételek szerint

Minősítsük a termékeket a beépített anyagok összértéke szerint!

A termék árát természetesen nemcsak az előállításához szükséges anyagok ára határozza meg, de a jelenlegi adatbázisban további információ nem áll rendelkezésre.

Legyen egy termék:

- *olcsó*, ha a beépített anyagok összértéke kisebb, mint 100 Ft;
- *közepes*, ha a beépített anyagok összértéke nagyobb vagy egyenlő, mint 100 Ft, és kisebb, mint 1000 Ft;
- *drága*, ha a beépített anyagok összértéke nagyobb vagy egyenlő, mint 1000 Ft.

I. Megoldás

Első lépésben termékenként számítsuk ki a termékbe beépített anyagok árát.

```
SELECT kód, SUM(mennyiség*egys_ár) AS aár
FROM szerkezet, anyag
WHERE szerkezet.azonosító = anyag.azonosító GROUP BY
kód;
AÁRAK{kód, aár}
```

Az AÁRAK táblából külön-külön gyűjtsük ki az olcsó, a közepes és a drága termékeket, majd az így kapott 3 táblát egyesítsük.

```
SELECT kód, 'olcsó'+Space(2)
FROM aarak
WHERE aár <100
UNIÓN
SELECT kód, 'közepes'
FROM aarak
WHERE aár >=100 AND aár <1000
UNIÓN
SELECT kód, 'drága'+Space(2)
FROM aarak
WHERE aár >= 1000;
```

Hajói adtuk meg a határokat a WHERE feltételben, minden termék egyszer és csak egyszer fog szerepelni az eredmény táblában, és az eredmény táblának ugyanannyi sora lesz, mint a TERMÉK táblának, mert minden terméket egyszer és csak egyszer besoroltunk a neki megfelelő kategóriába.

Az UNIÓN utasítással összeépített táblázatok oszlopainak kompatibiliseknek (azonos számúaknak, azonos típusúaknak és azonos hosszúaknak) kell lenniük, ezért egészítettük ki a konstansokat egy ACCESS függvény segítségével megfelelő számú betűközzel (az ACCESS nem követeli meg a kiegészítést).

II. Megoldás

Ha több kategóriába szeretnénk a termékeket besorolni, vagy a kategóriák határait időnként módosítani akarjuk, a fenti eljárás nagyon nehézkes. Általános elvként érdemes szem előtt tartani, hogy ne az SQL utasításokban tartsunk a módosítható adatokat.

A feladat megoldásához először a módosítható adatokból (a minősítéshez felhasznált adatokból) készítsük egy táblázatot! A legkisebb alsó határ legyen 0 (a beépített anyagok összértéke nem lehet negatív), a legnagyobb felső határhoz pedig válasszunk egy olyan nagy számot, ami az adatbázisban lévő termékeknel nem fordulhat elő.

MINŐSÍT

szöveg	alsó	felső
olcsó	0	100
közepes	100	1000
drága	1000	1000000000

A MINŐSÍT táblában az alsó és felső mező értékét bármikor módosíthatjuk, vagy bármikor újabb kategóriát hozhatunk létre egy új sor beírásával.

A termékek minősítése a MINŐSÍT tábla segítségével a következő:

```
SELECT kód, szöveg
FROM áarak, minősít
WHERE aár >= alsó
AND aár < felső;
```

Vegyük észre, hogy az AÁRAK és a MINŐSÍT táblák összekapcsolása nem kulcs - külső kulcs kapcsolat alapján történt! A MINŐSÍT táblának bármely oszlopa lehetne kulcs, de az AÁRAK táblának egyik oszlopában sem találhatók meg konkrétan a MINŐSÍT tábla értékei. Az összekapcsolás, az „értelmessé” tevés, a minősít.alsó, a minősít.felső és az áarak.aár között a lekérdezésben megfogalmazott összefüggés alapján történt.

Megszorítást jelent, hogy a MINŐSÍT tábla sorai között numerikus összefüggésnek kell lennie, amire a tábla feltöltésénél kell vigyáznunk, pl.: az olcsó termék felső határának egyenlőnek kell lennie a közepes termék alsó határával, különben egy terméknek lehet, hogy nem lesz megfelelő kategóriája, vagy lehet, hogy két kategóriának is meg fog felelni. Egy soron belül az alsó és felső mezők között is van összefüggés, alsó mező értéke kisebb, mint a felső mező értéke (úgy anis ha az alsó mező értéke nagyobb vagy egyenlő lenne, mint a felső mező értéke, annak az lenne a következménye, hogy ebbe a kategóriába egyetlen termék sem fog beletartozni). Ha van olyan termék, amibe beépített anyagok ára az utolsó sor felső értékénél nagyobb vagy egyenlő, akkor ez a termék nem fog szerepelni a besorolásban.

Csak akkor fogadhatjuk el a besorolást, ha meggyőződünk arról, hogy a fenti lekérdezés eredmény táblájának ugyanannyi sora van, mint a TERMÉK táblának, és az eredmény táblában minden termék kódja szerepel.

III. Megoldás

Alakítsuk át a MINŐSÍT táblát úgy, hogy hagyjuk el a felső oszlopot!

Az új tábla kulcsa az alsó mező legyen, aminek következtében az alsó mező értékei között nem szerepelhet két azonos érték. Természetesen az sem szerencsés, ha két kategóriának azonos lenne az elnevezése.

MINŐSÍT_2

s:	alsó
olcsó	0
közepes	100
draga	1000

III. 1. Megoldás

Egy SELECT utasításban a MINŐSÍT 2 tábla sorainak a sorrendjét nem tudjuk kihasználni. Egy termék abba a kategóriába tartozik, ahol a termékbe beépített anyagok összértéke nagyobb vagy egyenlő az alsó határnál, és nincs ennél az alsó határnál nagyobb alsó határ, amelyre ez az állítás igaz lenne.

```
SELECT kód, szöveg
FROM áarak, minősít_2 külső
WHERE aár >= külső.alsó
AND NOT EXISTS (SELECT alsó
FROM minősít_2 belső
WHERE belső.alsó > külső.alsó AND
aár >= belső.alsó);
```

(Az ACCESS a MINŐSÍT tábla alsó mezőjétől megköveteli, hogy ugyanúgy, mint az AÁRAK aár mezője, pénznem adattípus legyen, különben az aár >= belső.alsó feltételt nem hajtja jól végre! Az aár >= külső.alsó feltételt akkor is jól hajtja végre, ha a minősít.alsó egészszám. Az anyag.egys ár pénznem adattípusú volt, és az aarak.aár ezt örökölte. A CLng konverziós függvény használatával az aár mező értékét egészszámmá alakíthatjuk, és így a belső SELECT utasítás helyesen működik: CLng(aár) >= belső.alsó)

III. 2. Megoldás

Fogalmazhatjuk úgy is a besorolási feltételt, hogy az az alsó határ lesz a megfelelő, amely a legnagyobb a termékbe beépített anyagok összértékénél kisebb vagy egyenlő alsó határok közül.

```
SELECT kód, szöveg
FROM aarak, minősít_2 külső
WHERE aár >= külső.alsó
      AND külső.alsó = (SELECT MAX(belső.alsó)
                        FROM minősít_2 belső WHERE aár
                        >= belső.alsó);
```

Az optimalizáló tevékenységére enged következtetni egy ilyen lekérdezés eredménytáblájának a rendezettsége. A megjelenés olyan lesz, mintha a következő záradékkal fejeztük volna be az utasítást:

```
ORDER BY külső.alsó, kód;
```

Emlékeztetőül, az ACCESS SQL, ellentétben az SQL szabvánnyal, megengedi olyan mezőre is a rendezést, amelyiket nem jelenítettünk meg, de megjeleníthettünk volna.

A sorrendet nem befolyásolja, hogy a MINŐSÍT 2 táblának melyik mező a kulcsa. Ha a MINŐSÍT_2 táblának nem az alsó mező, hanem a szöveg mező lenne a kulcsa, a sorrend akkor sem változna. A belső SELECT utasítás eredményét a külső.alsó mezőhöz kell hasonlítani, és valószínűleg azért jelenik meg az eredmény ebben a sorrendben, mert így dolgozik a feldolgozó program a legkevesebbet. A belső SELECT utasítás paramétere az aarak.aar mező, azonban ennek az értéke nem befolyásolja a megjelenési sorrendet.

Vigyázzunk az összetettebb feladatoknál az „elvárt” sorrenddel! Különböző megvalósításokban nem egyformán működnek a végrehajtó programok. Ha az eredményt megadott sorrend szerint szeretnénk megjeleníteni, és nem vagyunk biztosak a feldolgozó program által biztosított megjelenítési sorrendben, vagy több különböző megvalósításban is végre akarjuk hajtani a lekérdezést, inkább használjuk az ORDER BY záradékot.

III. 3. Megoldás

Megoldhatjuk a feladatot úgy is, hogy először készítünk egy kétoszlopos lekérdezést. A lekérdezés eredménytáblájának az első oszlopában szerepelni fog minden termékkód, a második oszlopban pedig a MINŐSÍT_2 táblából az az alsó határ, amelyik a termékbe beépített összár alapján meghatározza, hogy melyik kategóriába tartozik a termék.

```
SELECT kód, MAX(alsó) AS határ
FROM minősít_2, áárak
WHERE alsó <= aár
GROUP BY kód;
```

KÓD_HATÁR{kód, határ}

Második lépésként a lekérdezés eredménytáblájában szereplő minden határhoz keressük meg a MINŐSÍT 2 táblából a határhoz tartozó minősítő szöveget.

```
SELECT kód, szöveg
FROM kód_határ, minősít_2
WHERE határ = alsó;
```

A két lekérdezés egy lekérdezéssé is összeépíthető.

```
SELECT kód, szöveg
FROM (SELECT kód, MAX(alsó) AS határ
      FROM minősít_2, áárak
      WHERE alsó <= aár
      GROUP BY kód),
minősít_2
WHERE határ = alsó;
```

A tábladefiniáló lekérdezés csak egyszer, a külső SELECT utasítás előtt, fog kiértékelődni.

A lekérdezés eredménytáblája, ellentétben az előző megoldással, kód szerint növekvő sorrendben fog megjelenni (a csoportosítás miatt lesz rendezett).

Ha azt szeretnénk, hogy a lista elején az olcsó, majd a közepes és végül a drága termékek termékkód szerinti sorrendben jelenjenek meg, akkor meg kell ad- - _mk a rendezési szempontokat is (nem a szöveg mező, hanem az alsó mezőnek reü lennie az első szempontnak):

```
SELECT kód, szöveg
      FROM (SELECT kód, MAX(alsó) AS határ FROM minősít_2,
                aárak WHERE alsó <= aár GROUP BY kód),
      minősít_2
 WHERE határ = alsó
 ORDER BY alsó, kód;
```

Vigyázat! A következő lekérdezés nem fogja a kívánt eredményt adni!

```
SELECT kód, MAX(alsó) AS határ, MAX(szöveg) rossz_szöveg
      FROM minősít_2, aárak WHERE alsó <= aár
      GROUP BY kód;
```

Az előzőekben láttuk, hogy hogyan tudjuk a kódhoz kiválasztani a határt, majd a határhoz a szöveget. Ebben a feladatban a határ mezőről és a rossz szöveg mezőről csak annyit tudhatunk, hogy az egy csoporton belüli értékek közül mindkettő külön-külön a legnagyobb, és ez kevés. A feladat megoldásához az kell, hogy a MINŐSÍT_2 táblában a határ és a rossz szöveg mező értéke ugyanabban a sorban szerepeljen, márpedig a fenti lekérdezésben ez csak véletlenül fordulhat elő.

Gyakorló feladatok

1. Soroljuk be a termékeket az eladott darabszám szerint. Sikeres egy termék, ha 1000 darabnál többet, sikertelen, ha 10 darabnál kevesebbet rendeltek meg belőle. A többit minősítsük közepesen sikertelennek.
2. Soroljuk be a termékeket az elmúlt hónapban megrendelt darabszám szerint. Sikeres egy termék, ha 1000 darabnál többet, sikertelen, ha 10 darabnál kevesebbet rendeltek meg belőle az elmúlt hónapban. A többit minősítsük közepes sikertelennek.
3. Soroljuk be a termékeket az elmúlt évben megrendelt darabszám szerint. Sikeres egy termék, ha az elmúlt év minden hónapjában 1000 darabnál többet, sikertelen, ha az elmúlt év minden hónapjában 10 darabnál kevesebbet rendeltek meg belőle. A többit minősítsük közepes sikertelennek.

21. Sávosan adott kedvezmény

1. Egy megrendelés összértékéből, annak a nagyságától függően, százalékos árengedményt adnak.

Számítsuk ki minden megrendelés összértékét és az engedmény nagyságát! 500.- Ft alatt nincs árengedmény. Az engedmény százalékos nagysága:

alsó határ	felső határ	százalék
500	- 1999	10%
	2000 - 20%	

I. Megoldás

Először számítsuk ki a megrendelések összértékét.

```
SELECT rend_száma, SUM(darab*ár) AS összár
FROM rendelés, termék
WHERE rendelés.kód = termék.kód
GROUP BY rend_száma;
```

-> R_ÖSSZ{rend_száma, összár}

Az engedményt legegyszerűbben az If függvénnyel számíthatjuk ki. (Az Access SQL a lista elemeinek az elválasztására a szabványos elválasztójeleket használja, függetlenül a Windows területi beállítástól!)

```
SELECT rend_száma, SUM(darab*ár) AS összár, If(összár<500, 0,
      If(összár<2000, 0.1*összár, 0.2*összár)) AS enged
FROM termék, rendelés
WHERE rendelés.kód = termék.kód
GROUP BY rend_száma;
```

II. Megoldás

Bár a fenti megoldás jó eredményt ad, két szempontból sem tekinthető hatékony megoldásnak. Egyrészt, ha változnak a kedvezmény kiszámításához szükséges adatok, magát az utasítást is át kell írni, másrészt, ha a kedvezmény sávok száma növekszik, az If függvény egyre bonyolultabbá válik.

Megoldás, ha létrehozuk a KEDVEZMÉNY{alsó, felső, százalék} táblázatot, és ezt felhasználva készítjük el a lekérdezést.

A fentiek alapján a KEDVEZMÉNY tábla:

KEDVEZMÉNY

alsó	felső	százalék
0	499	0
500	1999	0,1
2000	1000000	0,2

KEDVEZMÉNY {alsó, felső, százalék}

Természetesen a KEDVEZMÉNY tábla minden mezőjének értéket kell adni. Az alsó mező értékének egyvel nagyobbának kell lennie, mint a felső mező (rendezést feltételezve) előző sorban lévő értéke. Az alsó mező legkisebb értékének célszerű nullát választani. A felső határ utolsó elemének egy olyan nagy értéket célszerű beírni, ami biztosan nem fordulhat elő az adott rendszerben, legyen az adott rendszerben ez az érték 1000000.

A továbbiakban az egyszerűség kedvéért a RENDELÉS és a TERMÉK tábla helyett az I. megoldásban előállított R OSSZ lekérdezés eredménytábláját használjuk. A megoldás:

```
SELECT rend_szám,összár,összár*százalék AS enged
FROM r_össz,kedvezmény
WHERE összár >= alsó
AND összár <= felső;
```

III. Megoldás

Nem szerencsés a KEDVEZMÉNY tábla szerkezete.

Bár a KEDVEZMÉNY tábla egy sorában lévő mezők között csak annyi összefüggés van, hogy az alsó < felső, és ha ez az összefüggés nem teljesül, akkor is jó lehet még a lekérdezés, ha minden R OSSZ sorhoz találhatunk egy sort a KEDVEZMÉNY táblából, a sorok között azonban szoros kapcsolat van.

Az alsó-1 megadja az előző sor felső értékét, természetesen amennyiben van előző sor. A relációs adatbázis-kezelő rendszer a sorok közötti összefüggést nem tudja kezelni, számára nincs értelme előző és következő sornak. A 0 %-os sávot érdemes tárolni, hiszen ez is tartalmaz lényeges információt.

Készítsük el a KEDVEZMÉNY táblának redundanciától mentes részét, és tegyük el KEDV {határ, százalék} néven.

KEDV

határ	százalék
0	0
I 500	0,1
2000	0,2

-> KEDV{határ, százalék}

Az R OSSZ lekérdezés és a KEDV tábla alapján számoljuk ki megrendelésenként az engedmény nagyságát.

```
SELECT rend_száma,összár,összár*százalék
FROM r_össz,kedv
WHERE határ = (SELECT MAX(határ)
FROM kedv
WHERE határ <= r_össz.összár);
```

2. Hagyjuk el a KEDVEZMÉNY táblából a felső határokat (KED\ tábla), majd készítsünk egy lekérdezést a KEDV táblából, amelyik minden alsó határhoz kiszámítja a felső határt, azaz előállítja a KEDVEZMÉNY táblát!

A KEDV táblában nem szerepelt a KEDVEZMÉNY tábla felső mezőjének a legnagyobb értéke, az, amire csak a lekérdezés miatt volt szükségünk, és ami az adatbázisban a lehetséges legnagyobb megrendelés összértékénél nagyobb számot tartalmazott. Egészítsük ki a KEDV táblát egy sorral a következőképpen:

KEDV	
határ	százalék
0	0
500	0,1
2000	0,2
1000001	

Tekintsük a KEDV {határ, százalék} táblát. Feladat, hogy ebből állítsunk elő egy olyan eredménytáblát, amely azonos a KEDVEZMÉNY {alsó, felső, százalék} táblával.

A KEDVEZMÉNY táblában, mint az előző feladatnál láttuk, nyilvánvalóan van egy összefüggés. Minden felső határ számított mező, azonban az értékét nem az adott sorból, hanem, ha rendezve lenne a tábla, a következő sor alsó határ mezőjéből lehet kiszámítani.

A relációs adatmodellben egy táblának a következő sora értelmetlen fogalom. A „következő” sorról csak valamilyen rendezettség esetén beszélhetünk, ami nem a táblák, hanem a megjelenítés tulajdonsága. A relációs adatmodellben táblák vannak, a táblák soraira igazak, vagy hamisak a feltételek, és a táblák soraira vonatkozó feltételekkel kell meghatározni a táblák sorai között létező viszonyt, ami meghatározza valamilyen rendezettség szerinti „következő” sort. Olyan feltételt kell tehát megfogalmaznunk, amely a tábla minden sorára kiértékelési ismételtet a táblát, és az ismételt kiértékelés csak egy sorra, a „következő”

sorra lesz csak igaz. Ezzel a gondolatmenettel tudjuk az „előző” és a „következő” sort egymáshoz rendelni, azaz a sorrendet meghatározni.

I. Megoldás

Az eredménytáblának mind az alsó, mind a felső mezőjének a lehetséges értékei a KEDV tábla határ mezőjének az értékeiből származnak. Olyan lekérdezést kell tehát készíteni, ahol a KEDV táblát önmagával kapcsoljuk össze.

```
SELECT alsó_t.határ AS alsó, felső_t.határ AS felső  
FROM kedv AS alsó_t> kedv AS felső_t;
```

Az utasítás eredményeként létrejövő táblázatban a határ mező értékeinek minden lehetséges párosítása szerepelni fog. Biztosan nem kellenek ezen párok közül azok, ahol a felső határ nagyobb vagy egyenlő, mint az alsó határ.

```
SELECT alsó_t.határ AS alsó, felső_t.határ AS felső  
FROM kedv AS alsó_t, kedv AS felső_t WHERE alsó_t.határ <  
felső_t.határ;
```

Az így szűkített táblában egy adott alsó_t.határ-hoz azt a felső_t.határ-t kell hozzárendelni, amelyik a legkisebb. Csoportosítsuk ezért a táblát alsó_t.határ szerint, és minden csoportból válasszuk ki a legkisebb felső_t.határ-t tartalmazó sort. A felső határ az így kiválasztott minimum értéknél eggyel kisebb lesz, ugyanis a termékek ára és az összár is egész szám.

```
SELECT alsó_t.határ AS alsó, MIN(felső_t.határ) - 1 AS felső FROM  
kedv AS alsó_t, kedv AS felső_t WHERE alsó_t.határ <  
felső_t.határ GROUP BY alsó_t.határ;  
-> KEDV_1 {alsó, felső}
```

A KEDV táblából a százalék mezővel kiegészítve a KEDV_1 táblát megkapjuk a KEDVEZMÉNY táblát.

```
SELECT alsó, felső, százalék  
FROM kedv_1, kedv  
WHERE alsó = határ;  
KEDVEZMÉNY{alsó, felső, százalék}
```

A KEDV táblából közvetlenül is megkaphatjuk a KEDVEZMÉNY táblát, ha nemcsak az alsó_t.határ szerint csoportosítunk, hanem az alsó_t.százalék mező szerint is. A két szempont szerinti csoportosítás nem fog több csoportot eredményezni, mint amikor csak az alsó_t.határ szerint csoportosítottunk, mert minden alsó_t.határ szerinti csoportban csak egy százalékkérték fordulhat elő. Az alsó_t.határ egyértelműen meghatározza a százalék értékét.

```

SELECT alsó_t.határ AS alsó, MIN(felső_t.határ) - 1 AS felső,
       alsó_t.százalék AS százalék
FROM kedv AS alsó_t, kedv AS felső_t
WHERE alsó_t.határ < felső_t.határ GROUP BY alsó_t.határ,
       alsó_t.százalék;

```

Ugyanezt kaphatjuk akkor is, ha a második csoportosítás helyett az alsó t. százalék mező értéknek inkább a csoporton belüli maximumát (minimumát vagy átlagát) vesszük.

```

SELECT alsó_t.határ AS alsó, MIN(felső_t.határ) - 1 AS felső,
       MAX(alsó_t.százalék) AS százalék
FROM kedv AS alsó_t, kedv AS felső_t
WHERE alsó_t.határ < felső_t.határ
GROUP BY alsó_t.határ;
-> KEDVEZMÉNY{alsó, felső, százalék}

```

II. Megoldás

A kialakítandó KEDVEZMÉNY táblába az alsó és a százalék mező közvetlenül átvehető a KEDV táblából. A felső mező is átvehető a KEDV táblából, de ezt az értéket a táblának egy másik sorából kell átvenni, ezért a FROM záradékba kétszer vegyük fel a KEDV táblát.

```

SELECT alsó_t.határ AS alsó, felső_t.határ - 1 AS felső,
       alsó_t.százalék
FROM kedv AS alsó_t, kedv AS felső_t;

```

Ez a SELECT utasítás minden alsó_t-ből választott határhoz, hozzápárosít minden határt, amit a felső_t-ből választ ki. Minden párosításból csak egy értékpár jó. Minden alsóhoz ki kell választani az alsónál nagyobb határokból a legkisebbet, és ennél eggyel kisebb érték lesz a megfelelő felső határ. A megfogalmazásból látható, hogy a kiválasztásnál nem használtuk fel a KEDV tábla rendezettségét.

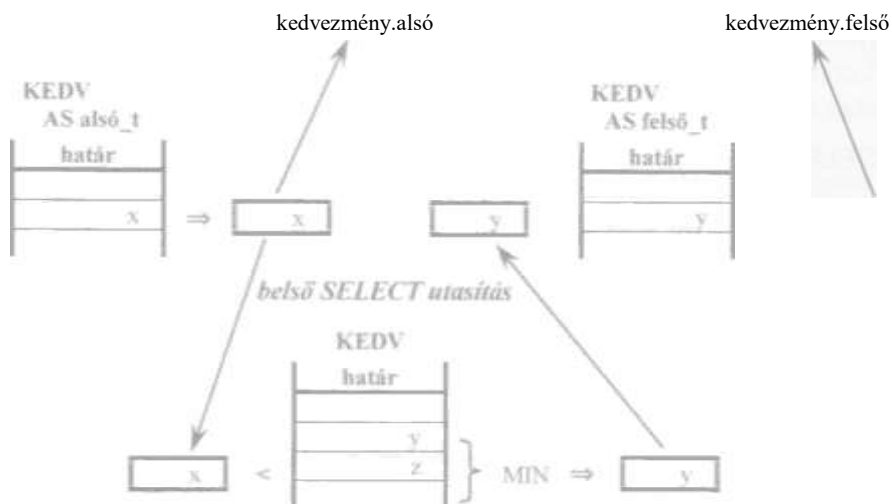
```

SELECT alsó_t.határ AS alsó, felső_t.határ - 1 AS felső,
       alsó_t.százalék
FROM kedv AS alsó_t, kedv AS felső_t
WHERE felső_t.határ =
      (SELECT MIN(határ)
       FROM kedv
       WHERE határ > alsó_t.határ);
       KEDVEZMÉNY{alsó, felső, százalék}

```

A külső SELECT utasításban kétszer használtuk fel a KEDV táblát, majd a belső SELECT utasításban még egyszer, soronként újra végig kell nézetni a KEDV táblát.

külső SELECT utasítás



III. Megoldás

A KEDV második táblájára - felső t - csak azért volt szükség, hogy a belső SELECT utasításban meghatározott határ értéket a külső SELECT utasításba át tudjuk adni. Az SQL szabvány megengedi, hogy a SELECT utasítás mezőlistájában is szerepelhessen beágyazott SELECT utasítás. Ezzel a megoldással a felső t tábla a SELECT utasításból kihagyható.

```
SELECT határ AS alsó,
       (SELECT MIN(határ)
        FROM kedv
        WHERE határ > alsó_t.határ) - 1 AS felső, százalék
FROM kedv AS alsó t;
```

A külső SELECT utasításban a tábla minősítésére szükség van, hogy a beágyazott SELECT utasításban hivatkozni tudjunk a külső tábla egy aktuális értékére, mivel mindkét SELECT utasításban ugyanazt a táblát használtuk a FROM záradékban.

Ennek a megoldásnak az eredménye egy sorral több, mint az előzőnek. Ez azért van, mert itt a KEDV tábla minden sorát kiíratjuk, az utolsót is. A KEDV tábla utolsó sorához, a legnagyobb értékű határ mezőhöz, a beágyazott SELECT utasítás egyetlen sort sem fog kiválasztani. Üres halmazon a MIN függvény értéke Null érték, ezért a beágyazott SELECT utasításnak a visszaadott értéke Null érték lesz. Ha ezt a sort, a határ mező legnagyobb értékéhez tartozó sort, nem akarjuk kiíratni, akkor erről gondoskodnunk kell. (Az előző utasításban a határ párokból válogattunk, így üres értékű felső határ nem fordulhatott elő.)

```
SELECT határ AS alsó,
      (SELECT MIN(határ)
       FROM kedv
       WHERE határ > alsóit.határ) - 1 AS felső,
      százalék
FROM kedv AS alsó_t
WHERE határ <> (SELECT MAX(határ)
               FROM kedv);
```

Ha a KEDV' tábla határ mezőjében lévő legnagyobb értékű mező sorában a százalék mező értékét üresen hagytuk (ezt az értéket sehol sem használtuk fel eddig), a nem kívánt sort elhagyhatjuk a százalék mező értékének a vizsgálatával is.

```
SELECT határ AS alsó,
      (SELECT MIN(határ)
       FROM kedv
       WHERE határ > alsó_t.határ) - 1 AS felső,
      százalék
FROM kedv AS alsó_t
WHERE százalék IS NOT NULL;
      KEDVEZMÉNY{alsó, felső, százalék}
```

IV. Megoldás

Vegyük észre, hogy az utolsó lekérdezés működni fog akkor is, ha a KEDV táblának az utolsó sora hiányzik, a KEDV tábla tehát legyen a következő:

KEDV	
határ	százalék
0	0
500	0,1 J
2000	

```
SELECT határ AS alsó,
      (SELECT MIN(határ)
       FROM kedv
       WHERE határ > alsó_t.határ) - 1 AS felső, százalék
FROM kedv AS alsó_t
WHERE százalék Is NOT Null;
```

A lekérdezés eredménytáblája csak annyiban különbözik a KEDVEZMÉNY táblától, hogy a felső mező a legnagyobb érték (1000000) helyett Null értéket tartalmaz, mert a beágyazott SELECT utasítás a határ mező legnagyobb értékét tartalmazó sorhoz nem választ ki egyetlen sort sem, így a MIN aggregáló függvény értéke Null érték, amiből egyet kivonva szintén Null értéket kapunk. Azt is érdemes megfigyelni, hogy a WHERE záradékra itt már nincs szükség.

```
SELECT határ AS alsó,
      (SELECT MIN(határ) FROM kedv WHERE határ >
       alsó_t.határ) - 1 AS felső, százalék
FROM kedv AS alsó_t;
```

Ha azt akarjuk, hogy pontosan a KEDVEZMÉNY táblát kapjuk meg, mivel a 1000000 érték nem szerepel¹ a KEDV táblában, akkor ezt az értéket a lekérdezésbe bele kell beírni.

```
SELECT határ AS alsó,
      If (határ = (SELECT MAX(határ) FROM kedv),
          1000000,
          (SELECT MIN(határ) FROM kedv WHERE határ >
           alsó_t.határ) - 1) AS felső, százalék
FROM kedv AS alsó_t;
```

Végezetül még egyszer felhívjuk a figyelmet arra, hogy egy elméleti felső határ beírása egy SQL utasításba, vagy egy adatbázis táblába nagyon veszélyes lehet, ha nem elég körültekintően járunk el. Inkább legyen egy kicsit bonyolultabb az SQL utasítás, de kerüljük az ilyen megoldásokat.

Gyakorló feladatok

1. Számítsuk ki 2001. májusában a megrendelések átlagértékét. Készítsünk egy táblázatot, amely szerint 20% kedvezményt adunk annak a megrendelőnek, aki a kiszámított átlagértéknél 50%-kal többért rendel meg, és 10% kedvezményt annak, aki 30%-kal többért. A táblázat tartalmazza a megrendelési értékhatárokat és a hozzájuk tartozó, adható kedvezmény százalékat. Kiinduláshoz használjunk fel következő táblázatot:

0	0
0.1	0.3
0.2	0.5

2. Számítsuk ki, hogy 2001. májusában melyik partner mennyi kedvezményt kapott volna, ha ez a lehetőség már érvényben lett volna.
3. Ha valaki egy termékből több darabot rendel meg, árengedményt kap. 10 darab termék felett egy paraméter által megadott százaléknyi árengedményt kap a megrendelő. Minden további 5 termék után az árengedmény a paraméter 10 százalékával nő, egészen 20 termékig (a határok tehát: 10, 15, 20). További megrendelésnél a kedvezmény mértéke nem változik. Készítsünk egy táblázatot, amelyik megmutatja egy további paraméterrel megadott termékekre, hogy hány terméktől hány termékig mennyi engedményt lehet kapni. Az engedményt a termék árára vetítve százalékban adjuk meg. Tervezzük meg az eredménytábla előállításához szükséges táblázatokat is!

22. Az eredménytábla sorainak sorszámozása

1. Számítsuk ki az egyes termékek bonyolultsági szintjét, amit az előállításához szükséges anyagfajták számával adunk meg! A bonyolultsági szint szerint adjunk a termékeknek egy folyamatosan növekvő sorszámot!

Először számítsuk ki a termékek bonyolultságát. A legkevésbé bonyolult terméknek, amelyik csak egyféle anyagból áll, a sorszáma legyen 1. Azonos bonyolultságú termékek közül az kapja a kisebb számot, amelyiknek a kódja kisebb. Az eredményt rendezzük a bonyolultsági mutató és azon belül a termék kódja szerint.

```
SELECT kód, COUNT(azonosító) AS bonyolultság
FROM szerkezet
GROUP BY kód
ORDER BY COUNT(azonosító), kód;
-> TERMÉK_B{kód, bonyolultság}
```

Az ORDER BY záradékot írhattuk volna a következő formában is, ahol a mezőkre az eredménytábla oszlopainak a sorszámmal hivatkozunk:

```
ORDER BY 2, 1;
```

Ha nem adtunk volna meg rendezési szempontokat, akkor termék.kód sorrendben jelentek volna meg a sorok, mert a csoportosítás erre a mezőre történt.

Az eredménytáblában a sorok pontosan abban a sorrendben jelennek meg, amilyen sorszámot kell kapniuk. A feladatot úgy is fogalmazhatjuk, adott egy táblázat, a sorai egy bizonyos rendezettségével, és a sorokat kell növekvő sorszámokkal ellátni.

A bonyolultság és a termékkód együttesen határozza meg a sorrendet. A sorszám a sorrendből következik. Az a probléma, hogy egy sor önmagában semmilyen információt nem tartalmaz arról, hogy a sor hol fog elhelyezkedni a rendezett táblában, ezt csak a sorok összessége határozza meg.

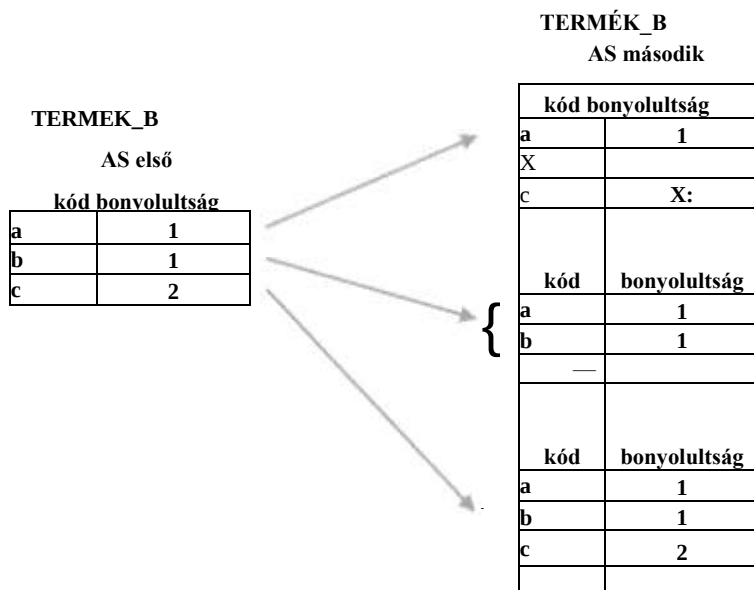
A megoldáshoz is ezt a gondolatot használhatjuk fel. Tekintsük az egész táblázatot. Egy adott sor azt a sorszámot fogja kapni, ahány sor van előtte a rendezettség szerint plusz még egy, önmaga.

Vegyük a táblát két példányban. Az elsőt használjuk arra, hogy a SELECT utasítással végigmegyünk minden során, a másodikat arra, hogy összeszámoljuk, hogy hány olyan sor van, amely a rendezettség szerint az adott sor előtt helyez-

kedik el. A gondolat jó, de nem felel meg az SQL logikájának. Ha belső SELECT utasítással kezeljük a második táblát, abból a konkrét sorszámot nem tudjuk visz- szaadni a külső SELECT utasításnak.

Finomítsuk az ötletet. Tekintsük a TERMÉK_B táblának önmagával alkotott direktszorzatát. Ebben a direktszorzatban minden egyes sorhoz annyi sor fog tartozni, ahány sora van a táblázatnak. Képzeletben tekintsük a két táblát kód és bonyolultság szerint rendezettnek. Az első tábla adott sorához a második tábla soraiból annyit őrizzünk meg, ahány sor van az első táblában előtte, plusz az önmagával alkotott párosítást. Az így kialakított táblázatban minden első táblabeli sor annyiszor fog szerepelni, amennyinek lennie kell a feladatban meghatározott sorszámanak.

```
SELECT első.kód, második.kód
FROM termék_b első, termék_b második
WHERE első.bonyolultság > második.bonyolultság
OR
(első.bonyolultság = második.bonyolultság
AND első.kód >= második.kód);
```



A táblázatot csoportosítsuk első.kód szerint, számítsuk ki, hogy egy csoportban hány sor van, és az lesz a sorszám. Rendezzük a táblát sorszám szerint, és ezzel megoldottuk a feladatot.

22. Az eredménytábla sorainak sorszámozása

```
SELECT COUNT(első.kód) AS sorszám, első.kód FROM
termék_b első, termék_b második WHERE
első.bonyolultság > második.bonyolultság OR
      (első.bonyolultság = második.bonyolultság
      AND első.kód >= második.kód)
GROUP BY első.kód
ORDER BY COUNT(első.kód);
```

Az eredménytábla használhatóságának van egy problémája. Hiányzik belőle az az információ, ami alapján a sorrendet eldöntöttük, hiányzik a bonyolultság.

Egy csoporton belül az első.bonyolultság mindig ugyanaz, hiszen ugyanabból a táblából származik, mint az első.kód. Ha tehát első.bonyolultság szerint is csoportosítunk, újabb csoport nem keletkezik, és akkor kiírhatjuk a bonyolultságot is.

```
SELECT COUNT(első.kód) AS sorszám, első.kód, első.bonyolultság
FROM termék_b első, termék_b második
WHERE első.bonyolultság > második.bonyolultság
      OR
      (első.bonyolultság = második.bonyolultság
      AND első.kód >= második.kód)
GROUP BY első.kód, első.bonyolultság ORDER BY 1;
```

Gyakorló feladatok

1. Változtassuk meg a bonyolultsági szint meghatározását! Most a legbonyolultabb terméknek, amelyik a legtöbbféle anyagból áll, a sorszáma legyen 1. Azonos bonyolultságú termékek közül az kapja a kisebb számot, amelyiknek a kódja nagyobb. (Fordított a sorszámozás, mint a kidolgozott feladatnál.)
2. Sorszámozzuk be a megrendelőket folyamatosan növekvő sorszámmal, aszerint, hogy mennyi volt a megrendelésük összértéke 2001. évben. Legyen 1 a sorszáma annak, aki a legtöbbet rendelt meg. Azok közül, akik azonos értéket rendeltek meg, annak legyen kisebb a sorszáma, akinek a partnerkódja kisebb. Rendezzük az eredménytáblát sorszám szerint növekvően!
3. Sorszámozzuk a megrendelőket hasonlóan az előző feladathoz, de most az azonos értéket megrendelő partnerek sorszáma legyen egyenlő (holtverseny megengedett). Rendezzük az eredménytáblát sorszám szerint növekvően, és azon belül, partnerkód szerint is növekvően.

23. Előre meghatározott sorú táblázat készítése

1. Készítsünk év közben az adott év megrendeléseinek az összegéről havi összesített listát! Az utolsó hónaphoz tegyünk egy megjegyzést, hogy ez a hónap még lezáratlan, még érkezhetnek megrendelések!

A szükséges információt három táblából tudjuk összeszedni: a RENDELÉSFEJ táblában van a megrendelés időpontja, a RENDELÉS tábla tartalmazza a megrendelt termékeket és a darabszámot, a TERMÉK tábla pedig a megrendelt termékek egységárát, tehát e három táblára lesz szükségünk. Egyesítsük a három táblát, utána szűkítsük, hogy az eredmény csak az ez évi megrendeléseket tartalmazza, majd csoportosítsuk a megrendelés hónapja szerint. Csoportosítás kifejezés szerint is lehetséges!

```
SELECT Month(rend_dátum) AS sorszám,  
       SUM(darab*ár) AS rend_érték, If(Month(rend_dátum) =  
       Month(Date()), 'lezáratlan', '')  
                                     AS lezáratlan  
FROM rendelésfej, rendelés, termék  
WHERE rendelésfej.rend_szám = rendelés.rend_szám  
  AND rendelés.kód = termék.kód  
  AND Year(rend_dátum) = Year(Date()) GROUP BY  
Month(rend_dátum);  
      HAVI_LISTA{sorszám, rend_érték, lezáratlan}
```

Az eredmény megfelel a feladatban leírtaknak. Az eredménytábla sorainak a száma nem lehet nagyobb, mint ahányadik hónapban vagyunk az adott évben, hiszen a rendelés dátuma nem lehet nagyobb, mint a mai nap. Kevesebb lehet, mert hiányozni fognak azok a hónapoknak megfelelő sorok, amelyekben nem volt semmilyen megrendelés. A GROUP BY záradék csak azokról a hónapokról készít összesítést, amelyik hónapban volt rendelés.

2. Készítsünk olyan listát, amelyik tartalmazza az adott év minden hónapját, és a megrendelések összesített értékét, függetlenül attól, hogy az adott hónapra érkezett-e megrendelés! Amennyiben az adott hónapra még nem érkezhetett be megrendelés, mert az a hónap még nem kezdődött meg, a rendelés összértéke legyen üres. Módosítsuk továbbá az előző listát úgy, hogy a befejezett hónapok mellé kerüljön kiírásra, hogy az a hónap már lezárt!

Pontosítsuk a feladatot! Olyan eredménytáblára van szükségünk, amelynek pontosan tizenkét sora van, első oszlop tartalmazza a beszámolási időszakot, a második a rendelések összértékét, a harmadik pedig egy megjegyzést.

A problémát az okozza, hogy a megrendelések között nincs olyan sor, amelyik a még meg nem kezdett hónapok dátumát tartalmazza, így, bárhogyan csoportosítjuk a megrendeléseket, nem lesz olyan eredménysorunk, amelyik a még meg nem kezdett hónapokhoz tartozik. Az SQL utasítások között van olyan utasítás is, amelyik egy meglévő táblához további sorokat illeszt hozzá, de most próbáljuk megoldani a feladatot azokkal az utasításokkal, amelyek már meglévő táblák sorainak a válogatásával, összevonásával foglalkozik. Az eredménytáblába új sorok beírására most nincs lehetőség.

Készítsünk egy tizenkét soros segédtáblát, amelyik a hónapok sorszámát és nevét tartalmazza.

HÓNAPOK

sorszám	hónap
1	január
2	február
12	december

Nem jó megoldás

```
SELECT Format(Year(Date()),"0000") &      & hónap AS dátum,
       rend_érték,
       If(hónapok.sorszám < Month(Date()),'lezárva',
         If (hónapok.sorszám = Month(Date()),
             'lezáratlan !!!',' ')) AS lezáratlan
FROM hónapok, havijista
WHERE havijista.sorszám = hónapok.sorszám;
```

Ennek a lekérdezésnek, ugyanúgy, mint az első feladat megoldásában, csak annyi sora van, ahány hónapban érkezett már megrendelés. Ha egy hónapban még nem érkezett megrendelés, de a későbbiekben még érkezhet, vagy, ha egy hónapban már befejeződött, és nem történt semmilyen megrendelés, akkor annak a hónapnak a sora hiányozni fog a listából. A HÓNAPOK és HAVI_LISTA tábla sorainak a közös részét jeleníti meg ez a lekérdezés.

I. Megoldás

A lekérdezést egészítsük ki a HÓNAPOK táblából azokkal a sorokkal, amelyek hiányoznak a HAVI LISTA táblából. A sorszám mezőre szükségünk van, mert a dátum mezőre nem rendezhetünk, az most nem dátum értéket tartalmaz, hanem egy karaktersorozatot.

```
SELECT hónapok.sorszám,
       Format(Year(Date()),"0000") & ' ' & hónap AS dátum,
       rend_érték AS rendelt_érték, If(hónapok.sorszám <
       Month(Date()),'lezárva', If (hónapok.sorszám =
       Month(Date()),
       'lezáratlan !!!', ' ')) AS lezáratlan
FROM hónapok, havijista
WHERE hónapok.sorszám = havijista.sorszám
UNION
SELECT hónapok.sorszám,
       Format(Year(Date()),"0000") & ' ' & hónap AS dátum, Null
       AS rendelt_érték, If(hónapok.sorszám <
       Month(Date()),'lezárva', If (hónapok.sorszám =
       Month(Date()),
       'lezáratlan !!!', ' ')) AS lezáratlan
FROM hónapok
WHERE hónapok.sorszám NÖT IN (SELECT sorszám
                               FROM havijista)
ORDER BY hónapok.sorszám;
```

Érdemes megfigyelni a két SELECT utasítás közötti különbséget!

II. Megoldás

A feladatot egyszerűbben is megoldhatjuk. A lekérdezésben a HÓNAPOK táblának a sorain menjünk végig, és a SELECT utasítás *mezőlistájában egy beágyazott SELECT utasítással válasszuk ki a HAVI LISTA táblából a megfelelő rendelési összértéket*. Ha a HAVI LISTA táblában nincs az adott hónapnak megfelelő sor, az eredménylistában a megrendelések összértéke helyén üres érték fog megjelenni, de maga a sor, az összes mezőjével, természetesen nem marad el.

```

SELECT Format(Year(Date()),"0000") +      & hónap AS dátum,
      (SELECT rend_érték
        FROM havijista
        WHERE hónapok.sorszám = havijista.sorszám)
      AS rendelt_érték,
      If(sorszám < Month(Date()),'lezárva',
        If (sorszám = Month(Date()),'lezáratlan !!!',' '))
      AS lezáratlan
FROM hónapok;

```

Az ACCESS karaktersorozatok összeépítésére mind a +, mind az & jelet megengedi műveletijeiként.

Vegyük észre, hogy az SQL utasítás végrehajtásával nem hoztunk létre új sort. A tizenkét soros HÓNAPOK tábla sorait listáztuk ki, szűkítő feltétel nélkül. Amennyiben a SELECT utasítás mezőlistájában lévő belső SELECT utasítás nem választ ki egyetlen sort sem, a rendelt érték mező értéke Null érték lesz, de ez a másik két mező értékét nem befolyásolja, a teljes sor kiírásra kerül. Az eredménylista HÓNAPOK tábla kulcsának, a sorszám mezőnek, növekvő sorrendjében fog megjelenni.

Gyakorló feladatok

1. Készítsünk egy listát, amelyik tartalmazza a „dda” partnerkódú megrendelő elmúlt ötévi megrendeléseink évenkénti összértékét! Ha valamelyik évben nem rendelt meg semmit, írjunk ki abba a sorba egy figyelmeztető megjegyzést!
2. Készítsünk egy lekérdezést, amelyikben paraméterrel lehet megadni egy termékkódot és a lekérdezés az elmúlt évre, havi bontásban megadja a termékből megrendelt darabszámot. Ha a termékből az adott hónapban nem rendeltek meg semmit, írjunk ki egy figyelmeztető megjegyzést az adott sorba! Használjuk a HÓNAPOK táblát.
3. Módosítsuk úgy az előző lekérdezést, hogy a lekérdezés az elmúlt 12 hónapra adja meg a termékből megrendelt darabszámot.

24. Számítások a SELECT utasítás mezőlistájában

1. A cég 10% kedvezményt ad az eredeti árból azoknak a megrendelőknek, akik az „A” betűvel kezdődő kódú termékekből két különböző terméket vásárolnak. További 10% kedvezményt kap az a vásárló, aki olyan 2 különböző terméket vásárol meg, amelyeknek az összértéke 10000 Ft felett van.

Készítsünk egy olyan lekérdezést, amelyik minden lehetséges kedvezményes termékpárról megadja az eredeti árat, a kedvezmény nagyságát és a kedvezményezett árat!

A kedvezmény csak az „A” betűvel kezdődő kódú termékekre vonatkozik. Két különböző terméket kell megvenni, és akkor kaphat valaki 10, illetve 20 % kedvezményt. A kedvezmény tehát termékpárookra vonatkozik. Először válasszuk ki a kedvezményes termékkört:

```
SELECT kód, ár
FROM termék
WHERE kód LIKE 'A*';
```

-> ABETŰS{kód, ár}

Készítsünk a kiválasztott termékkódokból egy olyan listát, amelyikben minden lehetséges pár szerepel.

A lista elkészítéséhez az ABETŰS eredménytáblának minden sorát minden sorával párosítjuk, azaz elkészítjük a táblázat önmagával alkotott direktszorzatát.

```
SELECT a1.kód, a2.kód
FROM abetűs a1, abetűs a2;
```

Jelöljük a kiválasztott termékkódokat ki, k2, ... kn szimbólumokkal, akkor a kód párokat egy kétdimenziós tömbben így ábrázolhatjuk:

{k1;k1}	{k1;k2}	{k1;k3}		{k1;kn}
{k2;k1}	{k2;k2}	{k2;k3}		{k2;kn}
{k3;k1}	{k3;k2}	{k3;k3}		{k3;kn}
{kn;k1}	{kn;k2}	{kn;k3}		{kn;kn}

Ha az ABETÜS eredménytáblának n sora volt, akkor ennek a tömbnek $n \times n$ eleme lesz, ami azt jelenti, hogy $n \times n$ sora lesz a fenti lekérdezésnek.

Ugyanabból a termékből alkotott pár nem felel meg annak a feltételnek, hogy két különböző terméket kell vásárolni, ezért hagyjuk ki ezeket az eredménytáblából.

```
SELECT al.kód, a2.kód
FROM abetűs a1, abetűs a2
WHERE al.kód <> a2.kód;
```

{k1;k1}	{k1;k2}	{k1;k3}		{k1;kn}
{k2;k1}	{k2;k2}	{k2;k3}		{k2;kn}
{k3;k1}	{k3;k2}	{k3;k3}		{k3;kn}
{kn;k1}	{kn;k2}	{kn;k3}		{kn;kn}

A sorok száma már csak $n \times (n-1)$ lett az eredménytáblában.

Ebben a listában minden pár kétszer szerepel. Benne van például a {k2;k1} és a {k1;k2} is. Csak azokat a párokat hagyjuk meg, ahol az első kód kisebb, mint a második kód.

```
SELECT al.kód, a2.kód
FROM abetűs a1, abetűs a2
WHERE al.kód <> a2.kód
AND al.kód < a2.kód;
```

A feltétel első része elhagyható:

```
SELECT al.kód, a2.kód
FROM abetűs a1, abetűs a2
WHERE al.kód < a2.kód;
```

{k1;k1}	{k1;k2}	{k1;k3}		{k1;kn}
{k2;k1}	{k2;k2}	{k2;k3}		{k2;kn}
{k3;k1}	{k3;k2}	{k3;k3}		{k3;kn}
{kn;k1}	{kn;k2}	{kn;k3}		{kn;kn}

A sorok száma végül: $\frac{n \times (n-1)}{2} = \frac{n(n-1)}{2}$

$$\frac{n!}{2! \cdot (n-2)!} = \frac{n!}{2}$$

(A sorok száma egyenlő azzal a számmal, amelyik megadja, hogy n különböző elem közül hányféleképpen lehet 2 elemet kiválasztani.)

Bonyolultabb, de az előző lekérdezéssel azonos eredményt adó lekérdezés:

```
SELECT al.kód, a2.kód
      FROM abetűs a1, abetűs a2
      WHERE a2.kód IN (SELECT kód
                      FROM abetűs
                      WHERE kód > al.kód);
```

A fentiek alapján a feladatnak megfelelő lekérdezés a kedvezményeket egészszámmra csonkítva az Int függvénnyel:

```
SELECT al.kód AS [első termék], a2.kód AS [második termék], al.ár
      + a2.ár AS [eredeti ár], Int([eredeti ár] * 0.1) AS kedvezmény,
      Int(If([eredeti ár] > 10000, [eredeti ár] * 0.1, 0)) AS [további
      kedv],
      [eredeti ár] - [kedvezmény] - [további kedv] AS [új ár] FROM
      abetűs a1, abetűs a2
      WHERE al.kód < a2.kód
      ORDERBY al.kód, a2.kód;
```

2. A cég 10% kedvezményt ad egy megrendelés végösszegéből azokra a megrendelésekre, amelyekben szerepel „A” betűvel kezdődő kódú termékekből legalább két különböző termék. A megrendelés végösszegéből további 10% kedvezményt kap az a megrendelő, akinek a megrendelésének a végösszege nagyobb, mint 10000 Ft. Ha nem vásárolt meg legalább kétféle „A” betűvel kezdődő kódú terméket, nem kap semmilyen kedvezményt.

Készítsünk egy lekérdezést, amelyik rendelésszámonként megadja a megrendelés kedvezmények nélküli végösszegét, a kétféle kedvezmény aktuális nagyságát, a teljes kedvezmény értékét és a kedvezmények figyelembevételével kialakított összértéket!

A második 10% kedvezmény akkor jár a megrendelőnek, ha a megrendelés összértéke nagyobb, mint 10000 Ft.

Induljunk ki abból, hogy 10% kedvezményt adunk arra a megrendelésre, aminek a végösszege 10000 Ft felett van. A továbbiakban a megrendeléseknek kedvezmények nélküli végösszegét nevezzük el eredeti árnak.

```
SELECT rend_száma,  
SUM(darab*ár) AS [eredeti ár],  
If([eredeti ár] > 10000, [eredeti ár]*0.9, [eredeti ár])  
AS [kedv ár]  
FROM termék, rendelés  
WHERE termék.kód = rendelés.kód  
GROUP BY rend_száma;
```

Az a gond, hogy így mindenki, aki 10000 Ft felett rendelt meg, kedvezményt kapott, pedig csak azoknak jár kedvezmény, akik egy rendelésszámmra kétféle „A” betűvel kezdődő kódú terméket is megrendeltek.

Gyűjtsük ki azokat a megrendeléseket, amelyeken legalább két különböző „A” betűvel kezdődő termék is szerepel, és tegyük el a KET_A táblába.

```
SELECT rend_száma  
FROM rendelés  
WHERE kód LIKE A*  
GROUP BY rend_száma  
HAVING COUNT(kód) >= 2;
```

-> KÉT_A{rend_száma}

A feladat így két részből áll. Kétféle megrendelés van. Egyrészt vannak olyanok, akik kapnak valamennyi kedvezményt, mert megrendeltek legalább kétféle „A” betűvel kezdődő kódú terméket, a másik csoport nem kap kedvezményt. Minden megrendelés beletartozik egyik és csak az egyik csoportba. Azt, hogy kap valaki kedvezményt, onnan tudjuk, hogy a megrendelésének a rend_száma benne van a KET_A táblában. A kétféle megrendelésre két külön lekérdezést készítsünk és egyesítsük UNIÓN paranccsal.

Áttekinthetőség kedvéért számítsuk ki külön a kedvezményt is. Mind a kedvezményt, mind a kedvezményezett árat egészszámmra kerekítsük. (Számolásnál a kedvezményezett árat számítsuk ki először, és azt kerekítsük egészszámmra a Round függvénnyel, majd ebből számítsuk ki a kedvezményt.)

```
SELECT rend_száma,  
SUM(darab*ár) AS [eredeti ár],  
[eredeti ár] - [kedv ár] AS [kedvezmény],  
Round(If([eredeti ár] > 10000,  
[eredeti ár]*0.8, [eredeti ár]*0.9), 0) AS [kedv ár]  
FROM termék, rendelés  
WHERE termék.kód = rendelés.kód  
AND rend_száma IN (SELECT rend_száma  
FROM két_a)  
  
GROUP BY rend_száma
```

UNIÓN

```

SELECT rend_száma,
       SUM(darab*ár) AS [eredeti ár],
       0,
       [eredeti ár]
FROM termék, rendelés
WHERE termék.kód = rendelés.kód
      AND rend_száma NOT IN (SELECT rend_száma FROM
                             két_a) GROUP BY rend_száma;

```

A két SELECT utasításra azért volt szükségünk, mert máshogy számoltuk a kedvezményezett végösszeget attól függően, hogy a rend_száma benne volt-e a KÉT_A táblában vagy sem. A kérdés eldöntésére a mezőlistában használhatjuk az If függvényt egy beágyazott SELECT utasítással. Számítsuk ki minden megrendelésnek az összértékét és keressük meg hozzá, hogy jár-e neki kedvezmény. Először vizsgáljuk meg, hogy megrendelt-e legalább két „A” betűvel kezdődő terméket, és ha igen, kap 10 % kedvezményt, majd, ha 10000 Ft felett rendelt meg, kap még 10 % kedvezményt az eredeti árból.

```

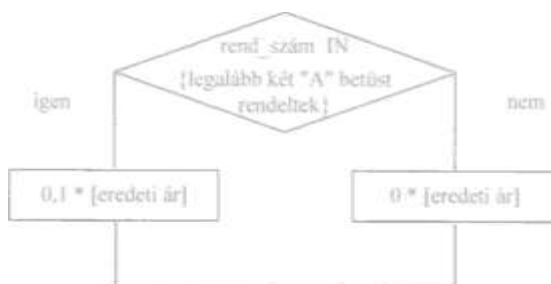
SELECT rend_száma,
       SUM(darab*ár) AS [eredeti ár], If(rend_száma IN (SELECT
rend_száma
FROM két_a),
0.1, 0) * [eredeti ár] AS [kedv 2A], If([kedv
2A] > 0 AND [eredeti ár] > 10000,
0.2* [eredeti ár], [kedv 2A]) AS [kedv ossz],
[eredeti ár] - [kedv ár] AS [kedvezmény], Round([eredeti ár]
- [kedv ossz], 0) AS [kedv ár] FROM termék, rendelés
WHERE termék.kód = rendelés.kód
GROUP BY rend_száma;

```

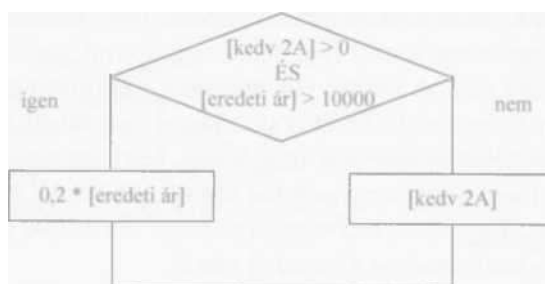
Figyeljük meg a következő oldali ábrán a SELECT utasítás mezőlistájának a kiértékelési sorrendjét. Minden rendelésre meghatározzuk a TERMÉK és RENDELÉS táblákból származó rend szám és eredeti ár értékeket, majd ezek segítségével számítjuk ki a kedv 2A, kedv ossz, kedvezmény, kedv ár mezőket.

A SELECT utasítás mezőlistájának nem kell azonos sorrendűnek lennie, mint a mezők kiszámításának a sorrendje, a példánk esetében nem is azonos. Természetesen átláthatóbbak a számítások, ha balról jobbra követik egymást a számított mezők.

[kedv 2A] :=



[kedv össz] :=



[kedv ár] :=

Round([eredeti ár] - [kedv össz]; 0)

[kedvezmény] :=

[eredeti ár] - [kedv ár]

Gyakorló feladatok

1. A megrendelés teljes összegéből adjunk 10 % engedményt, ha a megrendelő a megrendelés hónapjában már harmadszor rendel 10000.- Ft felett!
2. A megrendelés teljes összegéből adjunk 10 % engedményt, ha a megrendelő olyan termékből rendel meg legalább 2 darabot, amit a rendelés dátumához képest 6 hónap óta nem rendelt meg senki!
3. Annak a megrendelőnek, aki olyan termékből rendel meg legalább 2 darabot, amit a rendelés dátumához képest 6 hónap óta nem rendelt meg senki, adjunk ezekre a termékekre 10% engedményt. A többi termékre nem adunk kedvezményt.

25. Azonos anyagokból felépülő termék-p?rok kikeresése

Keressük meg azokat a termékeket (termékpárokat), amelyek azonos anyagokból épülnek fel!

I. Megoldás

Megjegyzések:

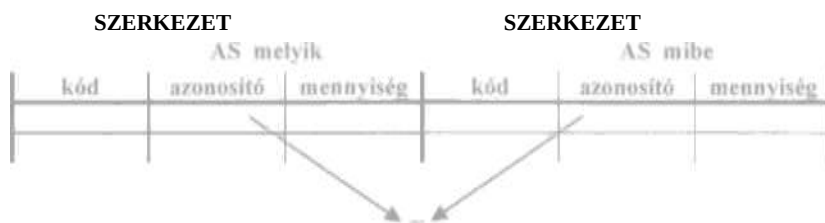
- az az információ, hogy egy termék miből épül fel, a SZERKEZET táblában van benne,
- azt kell megvizsgálni, hogy egy adott termékbe beépülő anyag milyen más termékekbe épül még be,
- ha az adott termékbe beépülő minden anyag szerepel egy másik termékben, és ez fordítva is igaz, akkor a két termék azonos anyagokból épül fel.

A SZERKEZET tábla tulajdonképpen a termékek és az anyagok közötti kapcsolatot írja le. Ezt a kapcsolatot két szempontból vizsgálhatjuk, egyrészt a termék oldaláról, hogy a termék milyen anyagokból áll, másrészt az anyag oldaláról, hogy az adott anyag milyen termékekbe épül be. Mindkét információ benne van a SZERKEZET táblában.

A lekérdezés érdekében logikailag duplázzuk meg a SZERKEZET táblát. Az elsőt arra használjuk fel, hogy megállapítsuk, egy adott termék milyen anyagokból épül fel, a második táblát pedig arra, hogy egy adott anyag milyen termékekbe épül be.

1.1. Határozzuk meg, hogy egy adott termékbe beépülő valamelyik anyag milyen más termékbe épül még be!

SELECT melyik.kód, melyik.azonosító, mibe.kód FROM szerkezet
melyik, szerkezet mibe WHERE melyik.azonosító =
mibe.azonosító;



SZERKEZET			SZERKEZET		
AS melyik			AS mibe		
kód	azonosító	menyiség	kód	azonosító	menyiség
AA1	112		BB2	112	
AA1	114		BB2	114	
BB2	112		AA1	112	
BB2	114		AA1	114	
AA1	112		AA1	112	
AA1	114		AA1	114	
BB2	112		BB2	112	
BB2	114		BB2	114	
BB2	115		BB2	115	

Ennek a táblának egy sora azt fejezi ki például, hogy az „AA1” kódú termékbe beépülő „112” azonosítójú anyag a „BB2” kódú termékbe is beépül.

Azoknak a soroknak a száma, amelyeknek az első mezőjében „AA1” kód szerepel és a harmadik mezőjében „BB2” kód szerepel, megegyezik a két terméket felépítő közös anyagok számával.

Azoknak a soroknak a száma, amelyeknek az első mezőjében „AA1” kód szerepel és a harmadik mezőjében is „AA1” kód szerepel, megegyezik a terméket felépítő anyagok számával.

1.2. Határozzuk meg, hogy két termék felépítésének hány közös anyaga van!

Az előző eredménytáblát csoportosítsuk melyik.kód és mibe.kód mező szerint:

```
SELECT melyik.kód, mibe.kód, COUNT(*) AS mennyi FROM
szerkezet melyik, szerkezet mibe WHERE melyik.azonosító =
mibe.azonosító GROUP BY melyik.kód, mibe.kód;
```

SZERKEZET SZERKEZET		
AS melyik		AS mibe
kód	kód	menyi
AA1	BB2	2
BB2	AA1	2
AA1	AA1	2
BB2	BB2	3

A két terméket felépítő anyagok közös részének a darabszámát adja az eredménytábla.

Konkrét példánkban az eredménytáblának van egy olyan sora, amelyikben AA1, BB2 és 2 érték szerepel. A sor jelentését értelmes mondattal így fogalmazhatjuk meg:

- az „AA1” kódú terméket felépítő anyagokból a „BB2” kódú termékbe kétféle épül be. Ebből következik, hogy lesz olyan sora is:
- a „BB2” kódú terméket felépítő anyagokból az „AA1 ” kódú termékbe kétféle épül be. Ha az „AA1” termék kétféle anyagból, a „BB2” háromféle anyagból épül fel, lesz a táblának olyan sora is:
- az „AA1” kódú terméket felépítő anyagokból az „AA1” kódú termékbe kétféle épül be,
- a „BB2” kódú terméket felépítő anyagokból a „BB2” kódú termékbe háromféle épül be.

Mindebből következik:

- az „AA1” kódú termék minden anyaga megtalálható a „BB2” termékben,
- de a „BB2” kódú terméknek van olyan anyaga, amelyik nincs a „AA1” termékben,
- azaz az „AA1” és a „BB2” kódú termék előállításához felhasznált anyagféslek nem azonosak.

Akkor épül fel két termék ugyanazokból az anyagokból, ha az adott termékpárra vonatkozó négy sorban a mennyi mező értéke egyenlő.

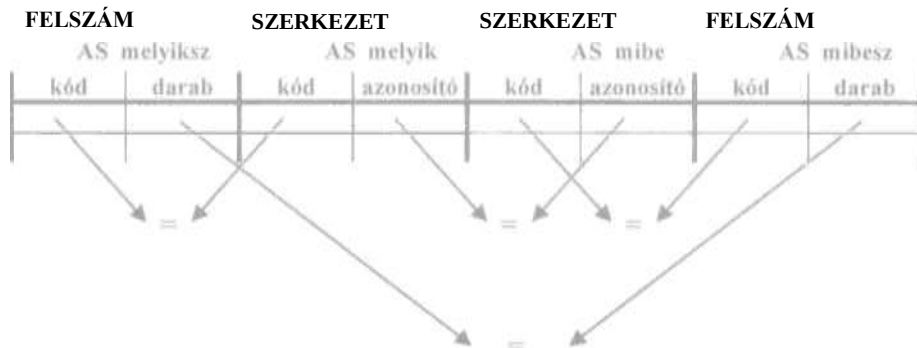
1.3. Számoljuk ki és tároljuk, hogy egy termék hányféle anyagból épül fel!

```
SELECT kód, COUNT(*) AS darab  
FROM szerkezet  
GROUP BY kód;
```

FELSZÁM{kód, darab}

1.4. Számítsuk ki, hogy két termékben hány közös anyag van! A termékpárokból hagyjuk ki azokat a termékeket, amelyekben a termékek különböző darabszámú anyagból épülnek fel!

Mind a melyik, mind a mibe táblához kapcsoljuk hozzá külön-külön a most kiszámolt FELSZAM táblát, és készítsünk el egy, a 2. lekérdezéshez hasonló új lekérdezést. A csoportosításba csak olyan termékkód párokat vegyünk figyelembe, amelyek azonos darabszámú anyagféslekből épülnek fel, azaz olyan sorokat, ahol a két figyelembevett FELSZAM táblában a darab mező értéke azonos.



```
SELECT melyik.kód, mibe.kód, COUNT(*) AS mennyi
FROM szerkezet melyik, felszám melyiksz, szerkezet mibe,
felszám mibesz
WHERE melyik.azonosító = mibe.azonosító
AND melyik.kód = melyiksz.kód
AND mibe.kód = mibesz.kód
AND melyiksz.darab = mibesz.darab GROUP BY
melyik.kód, mibe.kód;
```

Ebben a táblában már benne van a kérdésre adandó válasz, de sok felesleges sort tartalmaz.

Ez az eredménytábla mindig tartalmazni fogja az önmagával való párosítást, ezért a továbbiakban csak azokat válgassuk ki, ahol a melyik és a mibe kódja különböző.

Az eredménytábla minden termékpárt kétszer fog tartalmazni. Egyszer az egyik termékről megmondja, hogy a másik termékkel hány közös anyaga van, majd a másik termékről is elmondja ugyanezt.

Összegezve: csak olyan termékpárokból készített sorokat hagyjunk meg, ahol az elsőnek a kódja kisebb, mint a másodiké. Ezt elérhetjük vagy HAVING záradékkal, vagy a WHERE feltétel kiegészítésével, mivel mind az alaptáblákban, mind az egyesített tábla soraiban benne van a szűkítő információ.

```
SELECT melyik.kód, mibe.kód, COUNT(*) AS mennyi
FROM szerkezet melyik, felszám melyiksz, szerkezet mibe,
felszám mibesz
WHERE melyik.azonosító = mibe.azonosító
AND melyik.kód = melyiksz.kód
AND mibe.kód = mibesz.kód
AND melyiksz.darab = mibesz.darab
AND melyik.kód < mibe.kód
GROUP BY melyik.kód, mibe.kód;
```

1.5. Jelenítsük meg azokat a termékpárokat, amelyeknek minden anyaguk közös!

A fenti lekérdezésben a GROUP BY záradékkal képzett csoportokon belül mind a két FELSZAM táblából származó darab mező értéke azonos, ezért, ha bármelyikre további csoportosítást készítünk, nem jönnek létre újabb csoportok. A végső listába csak olyan termékpárok kelljenek, ahol a darab mező értéke megegyezik az adott csoport sorainak a számával.

```
SELECT melyik.kód, mibe.kód, COUNT(*) AS mennyi
      FROM szerkezet melyik, felszám melyiksz, szerkezet mibe,
           felszám mibesz
     WHERE melyik.azonosító = mibe.azonosító
           AND melyik.kód = melyiksz.kód
           AND mibe.kód = mibesz.kód
           AND melyiksz.darab = mibesz.darab
           AND melyik.kód < mibe.kód
     GROUP BY melyik.kód, mibe.kód, melyiksz.darab
     HAVING melyiksz.darab = COUNT(*);
```

Az első megoldás összegezve:

```
SELECT kód, COUNT(*) AS darab
      FROM szerkezet
     GROUP BY kód;

                                FELSZÁM{kód, darab}
```

Az 1.5. lekérdezésben a COUNT(*) oszlopra nincs szükségünk, elhagyhatjuk.

```
SELECT melyik.kód, mibe.kód
      FROM szerkezet melyik, felszám melyiksz, szerkezet mibe,
           felszám mibesz
     WHERE melyik.azonosító = mibe.azonosító
           AND melyik.kód = melyiksz.kód
           AND mibe.kód = mibesz.kód
           AND melyiksz.darab = mibesz.darab
           AND melyik.kód < mibe.kód
     GROUP BY melyik.kód, mibe.kód, melyiksz.darab
     HAVING melyiksz.darab = COUNT(*);
```

Bár az eredmény jó, de négy tábla összekapcsolásából készített lekérdezés nagy táblák esetén nagyon lassú.

II. Megoldás

A megfelelő párok kiválasztását két lépésben hajtjuk végre:

- először ne foglalkozunk a terméket felépítő anyagokkal, csak a felépítő anyagok számával, és keressük meg azokat a párokat, amelyek azonos darabszámú anyagféleségből állnak,
- utána ezekből válasszuk ki azokat, amelyeknél a felépítő anyagféleségek száma is azonos.

11.1. Számoljuk ki és tároljuk, hogy egy termék hányféle anyagból épül fel! (I. megoldás 3. pont)

```
SELECT kód, COUNT(*) AS darab
FROM szerkezet
GROUP BY kód;
```

FELSZÁM{kód, darab}

11.2. Jelenítsük meg azokat a termékkódpárokat, amelyek azonos darabszámú anyagfajtaból épülnek fel!

```
SELECT melyiksz.kód AS melyikkód, mibesz.kód AS mibekód,
melyiksz.darab
FROM felszám melyiksz, felszám mibesz
WHERE melyiksz.darab = mibesz.darab AND melyiksz.kód <
mibesz.kód;
-> AZONOSDB{melyikkód, mibekód, darab}
```

A párosításból az azonosakat elhagytuk (lásd: I. 4. megjegyzései)

Ennek a táblázatnak az előállításához használt direktzorzat lényegesen rövidebb lehet, mint az I. 4.-hez használté, ha a termékeket felépítő anyagok darabszáma sokféle.

11.3. Válogassuk ki azokat a termékkódpárokat, amelyek nemcsak azonos számú anyagféleségekből épülnek fel, hanem azonos anyagokból épülnek fel!

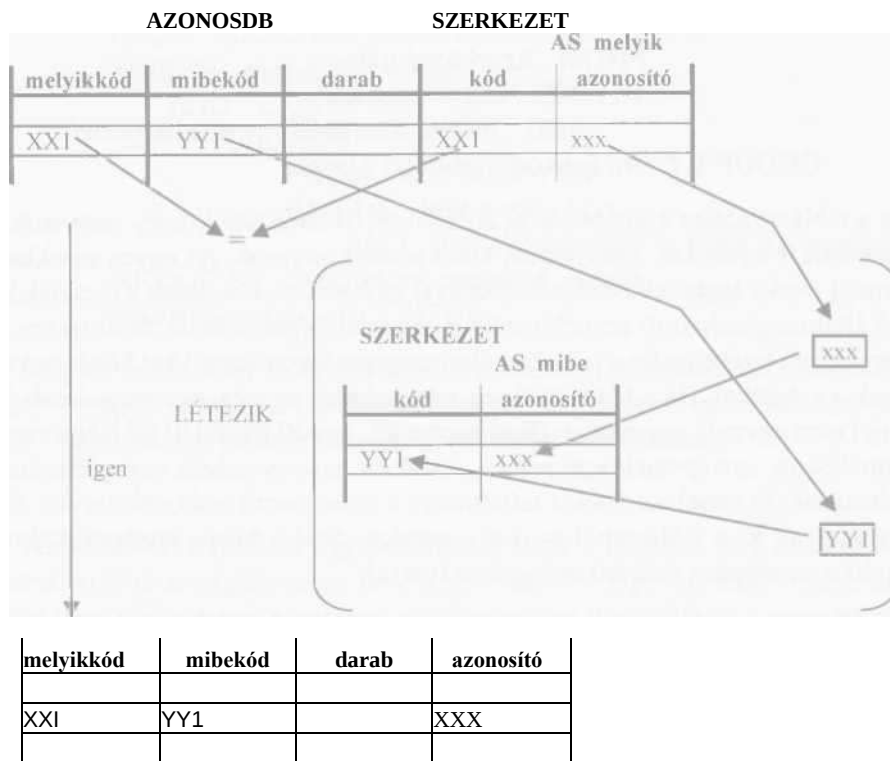
Először egészítsük ki az első termékkódot a termék szerkezetével.

```
SELECT melyikkód, mibekód, darab, azonosító
FROM azonosdb, szerkezet
WHERE melyikkód = kód;
```

Ez a táblázat annyiszor fogja tartalmazni a melyikkód mezővel meghatározott terméknek a teljes felépítését, ahány vele azonos darabszámú és nála nagyobb termékkódú termék létezik.

Válogassuk ki ebből a táblából azokat a sorokat, amelyeknek a harmadik mezőjében lévő kódhoz tartozó termék felépítésében szerepel a második mezőben lévő azonosító, így csak azok a sorok maradnak meg egy adott termék felépítésében, amelyek közös azonosítójú anyagot tartalmaznak.

```
SELECT melyikkód, mibekód, darab, azonosító
FROM azonosdb, szerkezet melyik
WHERE melyikkód = kód
AND EXISTS
(SELECT *
FROM szerkezet mibe WHERE kód = mibekód
AND melyik.azonosító = mibe.azonosító);
```



A belső SELECT utasítás a külső minden sorára kiértékelődik, mert tartalmaz külső hivatkozást (azonosdb.mibekód és melyik.azonosító). Ha a belső SELECT talál legalább egy megfelelő sort (többet nem is találhat), akkor EXISTS feltétel igaz lesz, a sor átkerül az eredménytáblába, különben a feltétel hamis lesz, és a sor kiesik.

Tekintsünk az AZONOSDB és a SZERKEZET tábla összekapcsolása után keletkező táblát. Ebben a táblában egy adott melyikkód és egy adott mibekód által meghatározott csoport sorainak a száma egyenlő a darab mező értékével. Ebből a csoportból annyi sor kerül át az eredménytáblába, amennyi a közös felépítő anyagaik száma. Ebből következik, hogy annak a két terméknek a felépítése azonos, amelyekhez tartozó sorok száma egyenlő a darab mezőben lévő értékkel.

Csoportosítsuk a tábla sorait a melyikkód és a mibekód mezők szerint, és számoljuk össze, hogy egy csoporton belül hány sor van. A csoporton belül a darab mező értéke azonos, ezért a csoportosításba azt is bevehetjük.

```
SELECT melyikkód, mibekód, darab, COUNT(*) AS mennyi
FROM azonosdb, szerkezet melyik
WHERE melyikkód = kód
AND EXISTS
  (SELECT *
   FROM szerkezet mibe
   WHERE kód = mibekód
    AND melyik.azonosító = mibe.azonosító)
GROUP BY melyikkód, mibekód, darab;
```

Ez a táblázat abban különbözik az AZONOSDB táblázattól, hogy nem tartalmazza azokat a párokat, amelyeknek nincs azonos anyaguk. Az egyes sorokban a mennyi értéke legfeljebb a darab értékével egyenlő, mert a darab a melyikkód mező által meghatározott termékben lévő anyagfélésegek számát tartalmazza, a mennyi mező értéke pedig a mibekód által meghatározott termékkel közös anyagoknak a számával. Ha a két terméknek nem minden anyaga azonos, a darab és mennyi nem egyenlő egymással. (Emlékeztetőül, az AZONOSDB táblában azok a termékpárok szerepelnek egy sorban, amelyek azonos számú anyagféléseget tartalmaznak, és ennek az értéket tartalmazza a darab mező értéke.)

Válogassuk ki a táblázatból azokat a sorokat, ahol a közös anyagok száma egyenlő a termékeket felépítő anyagok számával.

```
SELECT melyikkód, mibekód, darab, COUNT(*) AS mennyi
FROM azonosdb, szerkezet melyik
WHERE melyikkód = kód
AND EXISTS
  (SELECT *
   FROM szerkezet mibe WHERE kód = mibekód
    AND melyik.azonosító = mibe.azonosító)
GROUP BY melyikkód, mibekód, darab
HAVING darab = COUNT(*);
```

A második megoldás összegezve:

```

SELECT kód, COUNT(*) AS darab
  FROM szerkezet
 GROUP BY kód;

FELSZÁM{kód, darab}

SELECT melyiksz.kód, mibesz.kód, melyiksz.darab
  FROM felszám melyiksz, felszám mibesz
 WHERE melyiksz.darab = mibesz.darab
 AND melyiksz.kód < mibesz.kód;
    -> AZONOSDB{melyikkód, mibekód, darab}

SELECT melyikkód, mibekód
  FROM azonosdb, szerkezet melyik
 WHERE melyikkód = kód
    AND EXISTS
      (SELECT *
        FROM szerkezet mibe
       WHERE kód = mibekód
         AND melyik.azonosító = mibe.azonosító)
 GROUP BY melyikkód, mibekód, darab
 HAVING darab = COUNTf);

```

Vegyük észre, hogy az utolsó SELECT utasítás mezőlistájában a darab mező nem szerepel, azonban a GROUP BY záradékban a darab mezőnek szerepelnie kell, mert csak így hivatkozhatunk rá a HAVING záradékban.

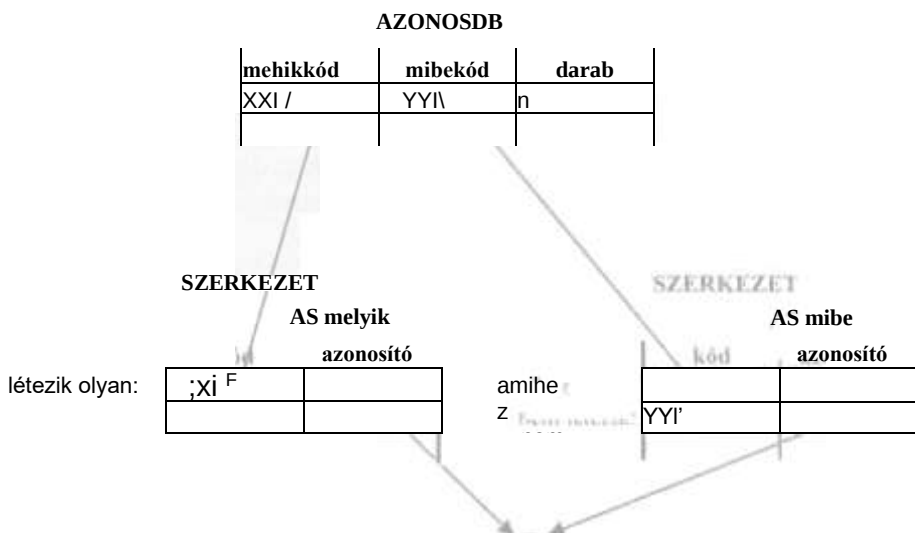
III. Megoldás

Az AZONOSDB táblát úgy készítettük, hogy a táblában lévő termékkódpárokat felépítő anyagféleségek darabszáma azonos. Egy termékkód párja akkor felel meg a feladatnak, ha az első minden anyaga megtalálható a másodikat felépítő anyagok között. Ebből következik, hogy a második minden anyaga is megtalálható az első felépítő anyagok között, mert mindkettő ugyanannyiféle anyagból áll.

Fordítva, akkor rossz a termékpár, ha van legalább egy olyan (azaz nem minden) az első felépítő anyagok között, amelyik nem található meg a másodikat felépítők között.

Más szóval, akkor rossz egy adott termékkódpár, ha létezik a SZERKEZET táblában olyan sor, amelyikben a kód egyenlő az első termékkóddal (ilyen sor annyi van, ahányféle anyagból áll az adott termék), és a hozzátartozó azonosító nem található meg a második terméket felépítő anyagok között.

Rossz a termékkódpaár: (melyikkód, mibekód) ha:



Jó egy termékkódpaár akkor, ha az előző mondat nem igaz. Azaz nem létezik a SZERKEZET táblában olyan sor, amelyikben a kód egyenlő az első termékkóddal, és nem létezik ehhez a sorhoz olyan sor a SZERKEZET táblában, amelyiknek a kódja egyenlő a második termékkóddal és az azonosító egyenlő a SZERKEZET táblából az előzőekben az első kód által meghatározott sorban található azonosítóval (ha van ilyen, akkor biztos, hogy csak egy van).

A harmadik megoldás:

```

SELECT kód, COUNT(*) AS darab
  FROM szerkezet
 GROUP BY kód;

FELSZÁM{kód, darab}

SELECT melyiksz.kód AS melyikkód, mibesz.kód AS mibekód,
       melyiksz.darab
  FROM felszam melyiksz, felszam mibesz
 WHERE melyiksz.darab = mibesz.darab
       AND melyiksz.kód < mibesz.kód;
    ■> AZONOSDB{melyikkód, mibekód, darab}
    
```

```
SELECT melyikkód, mibekód
FROM a7onosdb WHERE NÖT EXISTS (SELECT *
FROM szerkezet melyik
WHERE kód = melyikkód AND NÖT EXISTS
(SELECT *
FROM szerkezet mibe WHERE kód =
mibekód AND melyik.azonosító =
mibe.azonosító));
```

IV. Megoldás

A harmadik megoldás gondolatmenetét követve kiindulhatunk az eredeti TERMÉK táblából is.

Azonos anyagokból épül fel egy termékpár, ha az elsőt felépítő minden anyag azonosítója megtalálható a másodikat felépítő anyagok azonosítói között, és fordítva, ha a másodikat felépítő minden anyag azonosítója megtalálható az elsőt felépítő anyagok azonosítói között.

Keressük meg első lépésben azokat a termékpárokat, amelyeket felépítő minden anyag azonosítója megtalálható a másodikat felépítő anyagok azonosítói között. Ha ebben a táblázatban a második termék kódja is szerepelni fog úgy, hogy az őt felépítő minden anyag azonosítója megtalálható az elsőt felépítő anyagok azonosítói között, akkor a két termék azonos anyagokból áll. Természetesen olyan sorok is szerepelnek a táblában, ahol az első és második kód is azonos, ezeket szűljük ki.

Második lépésben válogassuk ki a termékkódpárokból azokat, amelyek fordított sorrendben is szerepelnek a táblában, és hagyjuk el az ismétléseket. Ezek lesznek a feladatnak megfelelő, minden anyagában azonos termékpárok.

```
SELECT melyik.kód AS melyikkód, mibe.kód AS mibekód
FROM termék melyik, termék mibe
WHERE melyik.kód <> mibe.kód
AND NÖT EXISTS
(SELECT *
FROM szerkezet melyiksz
WHERE melyiksz.kód = melyik.kód
AND NÖT EXISTS
(SELECT *
FROM szerkezet mibesz
WHERE mibesz.kód = mibe.kód
AND melyiksz.azonosító = mibesz.azonosító));
-> AZONOSBÓL{melyikkód, mibekód}
```

```
SELECT melyikkód, mibekód
FROM azonosból külső
WHERE melyikkód < mibekód
AND külső.melyikkód IN
      (SELECT belső.mibekód
      FROM azonosból belső
      WHERE külső.mibekód = belső.melyikkód);
```

Ez a megoldás az előzőektől abban különbözik, hogy ha léteznének olyan termékek, amelyeknek nem szerepel szerkezetleírásuk az adatbázisban, akkor ez a lekérdezés azokat is összepárosítja és visszaadja, mivel azonos a felépítő anyagféleségük. A feladatunk értelmében azonban ebben az adatbázisban minden terméknek kell lennie szerkezetleírásának, így ez a megoldás is ugyanazt az eredményt adja, mint az előzőek.

Gyakorló feladatok

1. Keressük meg azokat a partnerpárokat, akik az elmúlt évben ugyanazokat a termékeket rendelték meg!
2. Minden termékhez keressük meg azokat a termékeket, amelyekben bennlévő anyagok között legalább nyolcvan százalékban megtalálhatók a kiválasztott terméket felépítő anyagfajták!
3. Keressük meg azokat a partnereket, akik az elmúlt évnek legalább hat hónapjában mindig ugyanazokat a termékeket rendelték meg!

26. Különböző szerkezetű sorokból álló lista

1. Készítsünk egy listát, amely egy paraméterrel megadott termékről megjeleníti, hogy legyártható vagy sem! Írassuk ki, a termék kódját és nevét, valamint azt, hogy legyártható, vagy nem gyártható le!

A feladatnak több megoldását megtalálhatjuk aló. fejezetben. Megoldhatjuk például a következőképpen is:

```
SELECT termék.kód, név,  
       If(EXISTS  
         (SELECT szerkezet.kód  
          FROM szerkezet, anyag  
          WHERE szerkezet.azonosító = anyag.azonosító  
                AND mennyiség > készlet  
                AND szerkezet.kód = p_kód),  
         'nem gyártható le', 'legyártható') AS gyártható FROM  
termék  
WHERE termék.kód = p_kód;
```

2. Készítsünk egy listát, amely üres, ha egy paraméterrel megadott termék legyártható, és ha nem gyártható le, akkor a terméknek adja meg azokat az anyagait, amiknek a kis készlete miatt a terméket nem tudjuk legyártani!

```
SELECT kód, anyag.azonosító, neve  
FROM szerkezet, anyag  
WHERE szerkezet.azonosító = anyag.azonosító  
      AND mennyiség > készlet  
      AND kód = p_kód;
```

3. Egyesítsük a két listát, azaz egy termékről először állapítsuk meg, hogy legyártható vagy sem, majd, ha nem gyártható le, listázzuk azokat az anyagokat, amelyekből a készlet kevesebb, mint amennyi a termék legyártásához szükséges!

Az eredménynek tehát kétféle sorból kell állnia. Egyrészt a termékre vonatkozó sorban meg kell jelennie a termék kódjának, nevének és annak az információnak, hogy legyártható-e vagy sem, másrészt, amennyiben a termék nem gyárt

ható le, ki kell egészíteni a listát olyan sorokkal, amelyek azoknak az anyagoknak azonosítóját, nevét, készletét, mértékegységét, a legyártáshoz szükséges mennyiségét és a gyárthatósághoz szükséges hiányt tartalmazza, amelyek miatt nem lehet legyártani az adott terméket.

A feladatot két szerkezetileg különböző SELECT utasítás egyesítésével tudjuk megoldani.

Tervezzük meg először az eredménytáblát. Határozzuk meg, hogy az összetett táblázatban a két különböző lekérdezésből származó sorokból melyik mezők kerüljenek egymás alá, és mik legyenek az egyesített táblában a mezőnevek.

kód	név	gyártható	azonosító	neve	készlet	szükséges	hiányzik	mérték-egység
ELSŐ LEKÉRDEZÉS								
termék kódja	termék neve	"legyártható" vagy "nem gyártható le"	üres	üres	üres	üres	üres	üres
MÁSODIK LEKÉRDEZÉS								
üres	üres	"mert:"	anyag azonosítója	anyag neve	anyagból a készlet	egy termékhez szükséges mennyiség	egy termékhez mennyi hiányzik	anyag mérték-egysége

Az oszlopok számának és az egymás alatt lévő oszlopok típusának meg kell egyezniük, ezért az első SELECT utasításban üres oszlopokat kell definiálni. A második SELECT utasításban a mezők címének (az AS után megadott karaktereknek) nincs jelentősége. Azt, hogy a numerikus típusú üres oszlopban ne jelenjen meg a 0 érték, az oszlophoz rendelt formátummal vagy Null érték kiírásával lehet megoldani.

A megjelenő sorok megfelelő rendezettségéről is gondoskodjunk! A paraméterrel meghatározott termékre vonatkozó információt az első sorban a kell kiírni. Tekintettel arra, hogy a kód mezőnek csak az első sorban lesz ürestől különböző érték, ezért a rendezés első szempontja legyen a kód mező, csökkenő sorrendben. A második szempont az anyagazonosító növekvő sorrenddel legyen. Ezzel a rendezéssel elérhetjük, hogy elől lesz a termékre vonatkozó egyetlen sor, utána azonosító növekvő sorrendben a legyárthatóságot akadályozó anyagok. A rendezés az utasítás utolsó záradéka, és az egész eredménytáblára vonatkozik.

```

SELECT termék.kód, név,
       If(EXISTS
         (SELECT kód
          FROM szerkezet, anyag
          WHERE szerkezet.azonosító = anyag.azonosító
                AND mennyiség > készlet AND kód = p_kód),
         'nem gyártható le', 'legyártható') AS gyártható,
       Null AS azonosító, Null AS neve,
       Null AS készlet, Null AS szükséges,
       Null AS hiányzik, Null AS mértékegység
FROM termék
WHERE termék.kód = p_kód

```

UNIÓN

```

SELECT NullASx, Null AS y,
       mert:' AS gyártnató,
       anyag.azonosító, neve, készlet,
       mennyiség, mennyiség-készlet, mért_egys
FROM szerkezet, anyag
WHERE szerkezet.azonosító = anyag.azonosító
      AND mennyiség > készlet
      AND kód = p_kód
ORDER BY kód DESC, azonosító;

```

Az eredménytábla formázása már erősen rendszerfüggő, ezért ezzel itt nem foglalkozunk.

Gyakorló feladatok

1. Készítsünk egy paraméteres lekérdezést, amely a tárolt adatokból előállít egy „számlát” (hasznolt egy számlához)! Az eredménytáblának legyen külön fej- sora, és legyenek tételsorai.
2. Készítsünk egy listát, amelyik megjeleníti külön sorokban az „a” betűvel kezdődő termékek és külön sorokban az előállításukhoz szükséges anyagok minden adatát! A rendezettség biztosítása érdekében a lekérdezés utolsó oszlopában képezzünk a termék nevéből, kódjából és az anyag azonosítójából összeépített mezőt.
3. Készítsünk egy lekérdezést, amelyik a legnagyobb rendeiesszámú megrendelésnek az anyagszükségletét mutatja meg! A lekérdezés első sora a megrendelés adataiból a RFNDELÉSFEJ adatait jelenítse meg.

27. Összegfokozatos lista

1. Készítsünk egy listát, amely tartalmazza minden megrendelés rendelésszámát és a rendelés összértékét! A sorokban jelenítsük meg a megrendelés évét és hónapját!

A rendelések értékének a kiszámításához a RENDELÉS és a TERMÉK táblára van szükségünk. Azonban szükségünk van a rendelés évére és hónapjára is, amit RENDELÉSFEJ tábla rend dátum mezőjéből a Year és a Month ACCESS függvény segítségével számolhatunk ki. (Minden SQL környezetben vannak hasonló függvények.)

A három tábla „értelmes” (egyenlőségen alapuló természetes) összekapcsolása után Year(renddátum), Month(renddátum), rendelésfej.rend szám értékekre csoportosítva megkapjuk a kívánt lekérdezést. Csak emlékeztetőül, az eredménytáblának annyi sora lesz, ahány különböző rendelésszám van. A dátum a csoporton belül egyértelmű, dátum, illetve a dátum része szerinti csoportosítás új sorokat nem eredményez, csak azért szükséges, hogy értéküket az eredménytáblában meg tudjuk jeleníteni.

```
SELECT Year(rend_dátum) AS év, Month(rend_dátum) AS hónap,  
rendelésfej.rend_szám AS rendelésszám, SUM(darab*ár) AS  
[rendelt érték] FROM rendelésfej, rendelés, termék WHERE  
rendelésfej.rend_szám = rendelés.rend_szám AND rendelés.kód =  
termék.kód  
GROUP BY Year(rend_dátum), Month(rend_dátum),  
rendelésfej.rend_szám;
```

Az ACCESS megengedi a több szóból álló mezőneveket, de az összetartozást szögletes zárójelbe ([]) tevással kell jelölni.

2. Készítsünk egy listát, amely tartalmazza megrendelések összértékét havi bontásban! A sorokban jelenítsük meg a megrendelés évét és hónapját!

A feladat az előzőtől csak annyiban különbözik, hogy nem kell a rendelésszámot kiírni, így csoportosítani sem kell rendelésszámmra. Az eredménylistának annyi sora lesz, ahány különböző évben és hónapban történt megrendelés.

```
SELECT Year(rend_dátum) AS év,
       Month(rend_dátum) AS hónap, SUM(darab*ár) AS [rendelt
       érték] FROM rendelésfej, rendelés, termék WHERE
       rendelésfej.rend_szám = rendelés.rend_szám AND
       rendelés.kód = termék.kód
       GROUP BY Year(rend_dátum), Month(rend_dátum);
```

3. Készítsünk egy listát, amely tartalmazza évente a megrendelések teljes összértékét! A sorokban jelenítsük meg a megrendelés évét!

Ebben a lekérdezésben már a hónapokra történő csoportosításra sincs szükségünk. Az eredménylistának annyi sora lesz, ahány különböző évben történt megrendelés.

```
SELECT Year(rend_dátum) AS év, SUM(darab*ár) AS [rendelt érték]
FROM rendelésfej, rendelés, termék
WHERE rendelésfej.rend_szám = rendelés.rend_szám AND
       rendelés.kód = termék.kód
       GROUP BY Year(rend_dátum);
```

4. Készítsünk egy listát, amely tartalmazza a megrendelések teljes összértékét!

Itt már a csoportosítás is elmarad. Az eredménylistának csak egyetlen sora lesz.

```
SELECT SUM(darab*ár) AS [rendelt érték] FROM rendelésfej,
rendelés, termék WHERE rendelésfej.rend_szám = rendelés.
rend_szám AND rendelés.kód = termék.kód;
```

A lekérdezésben a rendelés dátuma sem szerepel, ezért a RENDELÉSFEJ táblát is kihagyhatjuk a lekérdezésből.

```
SELECT SUM(darab*ár) AS [rendelt érték] FROM rendelés, termék
WHERE rendelés.kód = termék.kód;
```

5. Az elkészült négy eredménytáblából, készítsünk egyetlen, összefokozatos listát!

Négy lekérdezést készítettünk el. Van egy tételes lista, egy havi összesítést, egy évi összesítést és a rendelésállomány összesített listája (egyetlen sor). Az a feladat, hogy az összesítéseket szűrjük be a megfelelő időszak után.

Az a probléma, hogy az SQL egy lekérdezéssel csak egynemű, azonos szintű sorokat tud az eredmény táblába összehozni. A különálló lekérdezésekből UNIÓN paranccsal és rendezéssel rakhatjuk össze a kívánt eredménytáblát.

Első eldöntendő kérdés, hogy az összesítő sorokat a részösszegek elé vagy után akarjuk kiírni. Válasszuk a következő formát:

- legyen fejléc is, ami csak az összesítés időszakát adja meg az összesítendő sorok előtt,
- legyen lábléc is, amely az összesített sorok után az összesítés időszakán kívül az összesített értéket is megadja.

A kétfajta sorból ugyanannyi van. Minden fejléchez tartozik egy lábléc. A kiíratandó fejléc és lábléc származhat ugyanabból az előre elkészített lekérdezésből. A/ eredménytáblában legyen egy külön oszlop, amelyik az összesítés szintjét adja meg. A lista tehát így nézzen ki:

táblanév amiből az adat származik	szint	időszak rendelésszá m	érték rendelt havi évi teljes
ÉV LISTA	ÉV eleje	év	
HÓNAP I ISTA REND_SZÁM_LISTA	HÓNAP eleje	év hónap év hónap rendelesszám	érték
REND SZÁM LISTA HÓNAP I ISTA	HÓNAP vége	év hónap rendelesszám év hónap	érték érték
HÓNAP LISTA REND SZÁM LISTA	HÓNAP eleje	év hónap év hónap rendelésszám	érték
REND SZÁM I ISTA HÓNAP LISTA	HÓNAP vége	év hónap rendelésszám év hónap	érték érték
HÓNAP LISTA	HÓNAP eleje	év hónap	
ÉV LISTA	ÉV vége	év	érték
ÉV LISTA	ÉV eleje	év	
HÓNAP LISTA REND SZÁM LISTA	HÓNAP eleje	év hónap év hónap rendelésszám	érték
REND SZÁM LISTA		év hónap rendelésszám	érték
HÓNAP LISTA	HÓNAP vége év hónap		érték
ÉVLISTA	ÉV vége év		érték
TELJES ÖSSZEG TEL'ES összeg			érték

Látszik a listaképből, hogy az ÉV LISTA és a HÓNAP LISTA táblára kétszer lesz szükségünk. A sorok megjelennek először az időszak elején érték nélkül, másodszor az időszak végén értékkel.

Már csak egy feladat van mielőtt az SQL utasításhoz hozzáfekszünk. Olyan rendezési ismérvet kell kialakítani az év, hónap és a rendelésszám mezőből, amelyik biztosítja a sorok helyét az eredménytáblában. Használjuk ki, hogy nincs 0. és 13. hónap, valamint a listát nem akariuk 5000-ben kiírni. Tegyük fel, hogy sem „0000000”, sem „ZZZZZZ7” rendelésszám nem létezhet. Amelyik lekérdezésre kétszer lesz szükségünk, abba tegyük el mindkét rendezési ismérvet.

Készítsük el a rendezési ismérvvel kiegészített táblázatokat. A lekérdezések elkészítésénél használjuk fel a már elkészített lekérdezéseket. Nem gazdaságos például az évi összesítést a napi forgalmi adatokból számítani, ha van már havi összesítés.

```
SELECT Format(év * 100 + hónap, "000000") &
rendelésfej.rend_száma AS rendezés, Year(rend_dátum) AS év,
Month(rend_dátum) AS hónap, rendelésfej.rend_száma AS
rendelésszám, SUM(darab*ár) AS [rendelt érték]
FROM rendelésfej, rendelés, termék
WHERE rendelésfej.rend_száma = rendelés.rend_száma
AND rendelés.kód = termék.kód
GROUP BY Year(rend_dátum), Month(rend_dátum),
rendelésfej.rend_száma;
-> REND_SZÁM_LISTA{rendezés, év, hónap, rendelésszám,
rendelt érték}
```

```
SELECT Format(év * 100 + hónap, "000000") & '0000000' AS
rendezési,
Format(év * 100 + hónap, "000000") & 'ZZZZZZ7' AS
rendezés2,
év, hónap,
SUM([rendelt érték]) AS rend_hónap
FROM rend_száma_lista
GROUP BY év, hónap;
HÓNAP_LISTA{rendezés1, rendezés2, év, hónap, rend_hónap}
```

```
SELECT Format(év * 100, "000000") & '0000000'
                                     AS rendezés3,
       Format(év *100 + 13, "000000") & 'ZZZZZZZ'
                                     AS rendezés4, év,
       SUM(rend_hónap) AS rend_év
FROM hónapjista GROUP BY év;
      ÉV_LISTA{rendezés3, rendezés4, év, rend_év}

SELECT Format(5000 * 100, "000000") & 'ZZZZZZZ' AS rendezés5,
       SUM(rend_év) AS rend_össz
FROM évjista;
      -> TELJES_ÖSSZEG{rendezés5, rend_össz}
```

A következő feladat a táblák egyesítése és rendezése. Felhasználva az előre megtervezett eredménytábla képét a lekérdezett mezők a megfelelő helyre kerülhetnek, ha a mezőlistákat jól állítjuk össze.

Az eredménytábla információtartalma a megtervezettől csak annyiban fog eltérni, hogy egy oszloppal több fog megjelenni, mert az első oszlop a rendezési ismérvet fogja tartalmazni.

```
SELECT rendezés, NULLÁS szint, év, hónap, rendelésszám, [rendelt
      érték], " AS [havi összesen], " AS [évi összesen], " AS
      [teljes összesen]
FROM rend_számlista
UNIÓN
SELECT rendezési, ' HÓNAP eleje', év, hónap,
      NULL, NULL,
      NULL, NULL, NULL
FROM hónapjista
UNIÓN
SELECT rendezés2, ' HÓNAP vége', év, hónap,
      NULL, NULL, rendjónap, NULL, NULL
FROM hónapjista
```

```
UNIÓN
SELECT rendezés3, 'ÉV eleje', év, NULL,
      NULL, NULL,
      NULL, NULL, NULL
FROM év_l ista
UNIÓN
SELECT rendezés-!, 'ÉV vége', év, NULL,
      NULL, NULL,
      NULL, rend_év,
      NULL
FROM évjista
UNIÓN
SELECT rendezésö, 'TELJES összeg', NULL, NULL,
      NULL, NULL, NULL, NULL, rend_össz
FROM teljes_összeg
ORDER BY rendezés;
```

Az eredménytábla formázása már túlmutat a példatár által felvállalt feladatkörön, bár nagyon fontos kérdés!

Természetesen az ilyen feladatokat nem feltétlenül az itt bemutatott módon kell megoldani! Minden rendszerben van professzionális listagenerátor (jelentéskészítő), amely biztosítja, hogy a felhasználó a feladatához legjobban illeszkedő megoldást megtalálja, az itt bemutatott módszernél sokkal kevesebb fáradtsággal. Itt csak azt akartuk megmutatni, hogy az ilyen típusú feladatokat is meg lehet oldani pusztán SQL lekérdezésekkel. Érdemes tudni, hogy az SQL környezetben működő listagenerátorok mögött is mindig SQL lekérdezések vannak.

Gyakorló feladatok

Nincsenek. Ilyen típusú feladatot jelentéskészítővel kell megoldani!

28. Külső ciklusváltozó szerinti válogatás

1. Ellenőrizzük, hogy az adatbázisban minden Aa- és Ab-vel kezdődő rendelésszám megvan-e, azaz létezik-e Aa-0000 és Ab-9999 közé eső minden rendelésszám! Jelenítsük meg a hiányzó számlaszámokat!

Első pillanatban a feladat meglepő, tekintettel arra, hogy a feladat olyan adatoknak a kigyűjtését kéri, amelyek nincsenek benne az adatbázisban. Márpedig az SQL nyelvnek a példatárban tárgyalt része meglévő táblázatok sorainak a kombinálásával és az így előállított sorokból történő válogatással foglalkozik.

Bármelyik hagyományos programozási nyelvben a feladatot egy egyszerű kétszintű ciklussal könnyen meg lehet oldani. A külső ciklus két lehetséges értéke „a” és „b”, a belső ciklust pedig 0-tól 9999-ig futtatva előállítható az összes olyan rendelésszám, amelyeknek meg kellene lenniük az adatbázisban. A feladat ennek alapján úgy fogalmazható, generálni kell a lehetséges rendelésszámokat, majd meg kell vizsgálni, hogy az így generált rendelésszámok benne vannak-e az adatbázisban.

Az SQL lekérdezéshez szükséges lenne egy olyan tábla, amelyik minden olyan rendelésszámot tartalmaz, ami a vizsgált intervallumba beleesik. Egy 20.000 soros tábla konkrét előállítását nyilvánvalóan szeretnénk elkerülni.

A megoldáshoz az az ötlet visz közelebb, hogy a 00-tól 99-ig terjedő számok előállíthatóak egyetlen 10 soros tábla önmagával képzet direktszorzataként.

Készítsünk el egy táblát, amelynek egyetlen egy karakteres oszlopa van: SZÁM {számjegy}. A táblának legyen 10 sora, és a mezők értéke 0, 1 ... 9 legyen. Ebből a táblából már könnyedén előállíthatjuk a 00-tól 99-ig terjedő karaktersorozatot.

```
SELECT első.számjegy & második.számjegy  
FROM szám első, szám második;
```

A lekérdezés eredménye 100 sor, amelyek rendre 00-tól 99-ig terjedő karakter-sorozatot fognak tartalmazni.

Térjünk vissza az eredeti feladathoz. A rendelésszám felépítése: Bb-9999, ahol B és b betű, a 9 számjegy. Készítsünk el három táblázatot. Mindháromban csak egy mező legyen, amely egy hosszúságú karaktersorozatot tartalmazzon:

```
ELSŐ{első_betű} tartalma: A;  
MÁSOD{második_betű} tartalma: a, b;  
SZÁM{szám_3_6} tartalma: 0, 1, 2 ... 9.
```

A következő lekérdezés egymásután generálja az Aa-val majd Ab-vel kezdődő összes lehetséges rendelésszámot, és megvizsgálja, hogy ezek benne vannak-e a RENDELESFEJ tábla rend szám oszlopának a mezőiben. Ha egy generált rendelésszámot nem talál meg, kiírja azt az eredmény táblába. Végül az így kapott hiányzó rendelésszámokat növekvő sorrendbe rendezi.

```
SELECT első_betű & második_betű & három.szám_3_6
      & négy.szám_3_6 & öt.szám_3_6 & hat.szám_3_6
      AS hiányzik
FROM első, másod,
      szám_három, szám_négy, szám_öt, szám_hat WHERE
első_betű & második_betű & három.szám_3_6 &
négy.szám_3_6 & öt.szám_3_6 & hat.szám_3_6
      NŐT IN (SELECT rend_száma
              FROM rendelésfej)
ORDER BY első_betű & második_betű &
      három.szám_3_6 & négy.szám_3_6 &
      öt.szám_3_6 & hat.szám_3_6;
```

Vegyük észre, hogy a keresett rendelésszámokat tartalmazó, 20.000 soros táblát nem mi készítettük el! Mivel a WHERE záradékban nem adtunk meg semmilyen összekapcsoló feltételt a FROM záradékban megadott táblákhoz, az SQL minden tábla minden elemét összepárosítja, és ezzel előállítja a szükséges táblát. A FROM záradékban előállított tábla sorainak a száma:

$$1 \times 2 \times 10 \times 10 \times 10 \times 10 = 20.000$$

A három segédtábla tartalmát egy újabb lekérdezés előtt bármikor módosíthatjuk, így tetszőszerinti rendelésszám intervallum teljességét megvizsgálhatjuk.

Gyakorló feladatok

1. Jelenítsük meg azokat az „l”-es karakterrel kezdődő háromjegyű számokat, amelyekhez nem tartozik anyagazonosító!
2. Jelenítsük meg azokat az „A” karakterrel kezdődő lehetséges termékkódokat (2 nagybetű és 1 számjegy), amelyek nem jelölnek létező termékeket!
3. Keressük meg, hogy az elmúlt évben melyik nap nem volt megrendelés! Hagyjuk ki az eredménytáblából a szombati és vasárnapi napokat!

29. Hiányzó értékek megkeresése folytonos és nem folytonos mezőértékekből

1. Folytonos rendelésszámokat feltételezve keressük meg a rendelésfej táblából a hiányzó rendelésszám-intervallumokat! (például rontott, sztorózott rendelések miatt hiányzik a rendelésszám)

A feladat megoldásánál az jelent problémát, hogy a relációs adatbázis-kezelésnek a könyvben tárgyalt része a létező táblázatokból vagy azok egyesítéséből sorokat válogat ki. A hagyományos adatfeldolgozással ellentétben, nem tud a lehetséges rendelésszámokra egy ciklust generálni, és így nem tudja a generált ciklusváltozónak és a létező rendelésszámoknak az összevetésével megállapítani, hogy melyek a hiányzó rendelésszámok.

A megoldáshoz az az ötlet fog elvezetni, hogy minden rendelésszámról megvizsgáljuk, hogy megvan-e az adatbázisban a nála eggyel nagyobb, illetve eggyel kisebb rendelésszám is. A szakadásokat, a rendelésszámban bekövetkező ugrásokat fogjuk megkeresni.

Először gyűjtjük ki az összes rendelésszámot a későbbi felhasználás céljából. Azért, hogy a rendelésszámmal műveleteket tudjunk végezni, bontuk szét két részre, egyrészt a rendelésszám betű részére, másrészt a rendelésszám szám részére. Használjuk fel az ACCESS Asc függvényét, amely egy karakternek a karakterkódját adja vissza, valamint Chr függvényét, amely egy karakterkódnak megfelelő a karaktert ad vissza eredményként.

```
SELECT rend_szám,  
       Left(rend_szám,2) AS betű, CInt(Right(rend_szám,4)) AS  
szám, If(Right(betű, 1 )='a',  
        Chr(Asc(Left(betű,1))-1) & 'z', Left(betű,1) &  
        Chr(Asc(Right(betű,1))-1)) As e_betű, If(Right(betű, 1 )='z',  
        Chr(Asc(Left(betű,1))+1) & 'a', Left(betű,1) &  
        Chr(Asc(Right(betű,1))+1)) AS k_betű FROM rendelésfej;  
-> RENDELÉSSZÁM{rend_szám, betű, szám, e_betű, k_betű}
```

Hiányzó rendelésszám ott van, ahol a rendezett rendelésszámokban ugrás van. Meg kell határozni a folyamatos rendelésszámok első és utolsó értékét. A folyamatos rendelésszámok első értékére az jellemző, hogy nincs nála eggyel kisebb sorszámu és vele egyenlő betűjelű rendelésszám.

A gondot az okozza, hogy az SQL-ben értelmetlen beszélni az előző, következő sorról, azaz nem tudjuk kihasználni a lekérdezésben a tábla rendezettségét.

Tekintsük a RENDELÉSSZÁM táblázat rend szám szerint rendezve, és helyezzük el a táblát egymás mellett kétszer:

		RENDELÉSSZÁM AS külső	RENDELÉSSZÁM AS belső
ugrás	folyamatos	Aa-x felső, mert:	Aa-x nem létezik Aa-x - 1
	U cd Z)	Bb-y nem felső, mert:	Bb-y - 1 Bb-y létezik Bb-y - 1
ugrás	->	Bc-0000 nem felső, mert:	Bb-9999 Bc-0000 létezik Bb-9999
	CA o	Cc-z felső, mert:	Cc-z nem létezik Cc-z - 1
	<2		

Gondoljuk végig, hogy betűváltás esetén mi történik! Ha rendelésszám szám része nagyobb nullánál, akkor a betűrész kiegészítve a számrcsz - 1 nyilván nem létezik, a sorozatban tehát ugrás van. Ha számrész nulla, akkor folyamatos a rendelesszám. ha létezik az ábécé szerinti előző betűpár kiegészítve 9099 karaktersorozattal.

SQL segítségével megkeresve a felső határokat, amikor tehát a rendezettséget nem használjuk ki:

```

SELECT rend_száma, betű, szám , e_betű FROM rendelésszám AS
külső WHERE NOT EXISTS
(SELECT belső.szám
FROM rendelésszám AS belső WHERE külső.szám
<> 0
AND belső.szám = külső.szám-1
AND belső.betű = külső.betű
OR külső.szám = 0
AND belső.szám = 9999 AND belső.betű =
külső.e betű);
FELSŐ_HATÁR{rend_száma, betű, szám, e_betű}
    
```

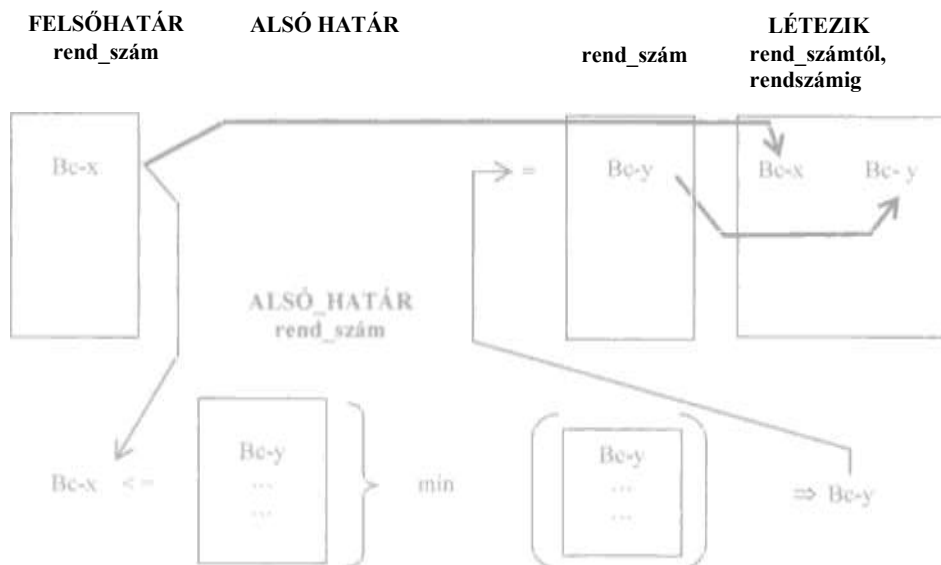
Ugyanígy meghatározható a csoportonkénti utolsó rendelésszám a folyamatos rendelésszámok közül:

		RENDELÉSSZÁM AS külső	RENDELÉSSZÁM AS belső
	folyamatos		
ugrás		Aa-p alsó, mert:	Aa-p nem létezik Aa-p+1
	<> O E >	Bb-q nem alsó, mert:	Bb-q Bb-q + 1 létezik Bb-q+1
		Bb-9999 nem alsó, mert:	Bb-9999 Bc-0000 létezik Bc-0000
ugrás	----->		
	° í		
	£	Cc-r alsó, mert:	Cc-r nem létezik Cc-r+1

```

SELECT rend_szám, betű, szám , k_betű
FROM rendelésszám AS külső
WHERE NOT EXISTS
    (SELECT belső.szám
    FROM rendelésszám AS belső
    WHERE külső.szám <> 9999
      AND belső.szám = külső.szám + 1
      AND belső.betű = külső.betű
    OR külső.szám = 9999
      AND belső.szám = 0
      AND belső.betű = külső.k_betű);
    ■> ALSÓ_HATÁR{rend_szám, betű, szám, k_betű}
    
```

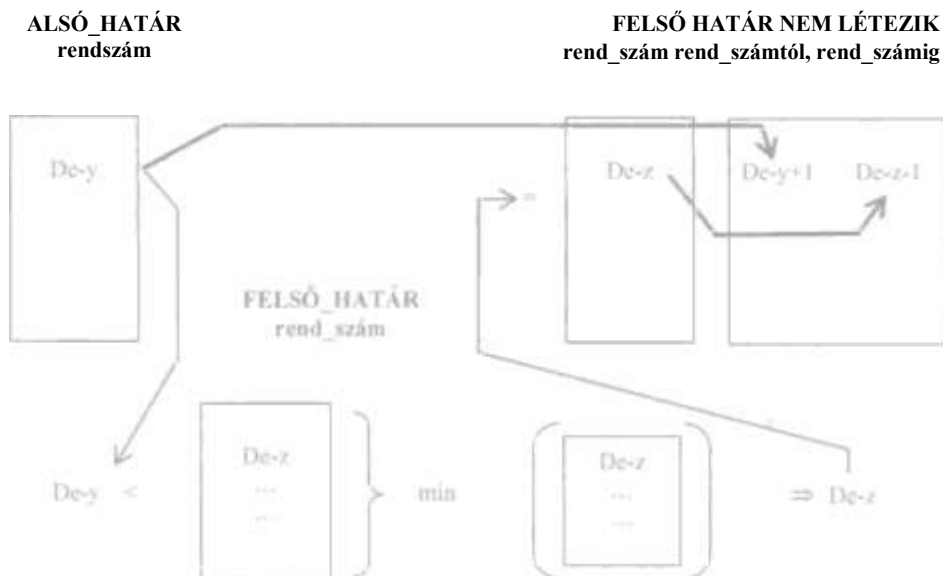
A folyamatos rendelésszámok az összetartozó felső és alsó határok között vannak. Egy adott felső határhoz az az alsó határ tartozik, amelyik az adott felső határnál nagyobb vagy egyenlő alsó határok közül a legkisebb:



Ha azt is ki akarjuk számítani, hogy egy intervallumban hány rendelésszám van, akkor figyelembe kell venni a betűváltásokat is. A rendelésszám betű része 2 betűből áll, és az angol ábécének 26 betűje van. A rendelésszám szám része 0- tól 9999-ig terjedhet.

```
SELECT felső_határ.rend_száma AS rend_számtól,
       alsó_határ.rend_száma AS rend_számgig,
       ((Asc(Left(alsó_határ.betű,1)) -
         Asc(Left(felső_határ.betű,1))) * 26 +
         Asc(Right(alsó_határ.betű,1)) -
         Asc(Right(felső_határ.betű,1))) * 10000 +
       alsó_határ.szám - felső_határ.szám + 1 AS darab FROM
       felső_határ, alsó_határ WHERE alsó_határ.rend_száma =
       (SELECT MIN(alsó_határ.rend_száma)
        FROM alsó_határ WHERE felső_határ.rend_száma <=
        alsó_határ.rend_száma);
       -> LÉTEZIK{rend_számtól, rend_számgig, darab}
```

Hasonló módon határozhatóak meg a hiányzó rendelésszám-intervallumok is. A lekérdezés minden alsó határhoz egy felső határt rendel hozzá:



```
SELECT If(alsó_határ.szám = 9999, k_betű && '0000',
        alsó_határ.betű & '-' &
        Format(alsó_határ.szám+1,'0000'))
        AS rend_számtól,
        If(felső_határ.szám = 0, e_betű & '-' & '9999',
        felső_határ.betű & '-' &
        Format(felső_határ.szám-1,'0000'))
        AS rend_számgig FROM
        alsó_határ, felső_határ WHERE felső_határ.rend_számgig =
        (SELECT MIN(felső_határ.rend_számgig)
        FROM felső_határ
        WHERE felső_határ.rend_számgig >
        alsó_határ.rend_számgig);
        NEM_LÉTEZIK{rend_számtól, rend_számgig}
```

Természetesen a hozzárendelést fordítva is végre lehet hajtani. Minden felső határhoz megkereshetjük a hozzá tartozó alsó határt.

2. Nem szükségképpen folyamatos rendelésszámokat feltételezve, keressük meg a hiányzó rendelésszámokat, és határozzuk meg ezeknek a darabszámát!

Ha a rendelésszámok sorozata nem folyamatos, a konkrétan hiányzó rendelésszámokat, csak akkor tudjuk meghatározni- ha van információnk arra vonatkozóan, hogy melyek a lehetséges rendelésszámok. A NEMLÉTEZIK lekérdezésben a nem létező rendelésszámoknak a darabszámát, hasonlóan a LÉTEZIK lekérdezésben bemutatott módszerhez, ki lehet számítani, de nem érdemes, mert esetleg a rendelésszám betűrészből egy egész betűcsoport teljesen hiányozhat.

A lehetséges rendelés számokat tartsuk nyilván egy sorszámozott relációban, amely a rendelésszám betűrészához megadja a számrész lehetséges alsó és felső határát. Legyen ez a reláció: LEHET {sorszám, betű, tói, ig}. Ebben a relációban minden betűpárt külön sorban kell megadni, és lehet, hogy egy betűpárhoz több sor tartozik. (Érdemes elgondolkodni azon, hogy ebben a relációban milyen szabályszerűségek vannak!)

Tegyük fel, hogy a RENDELÉSSZÁM {rend szám, betű, szám, e_betű, k betű} relációban minden rendelésszám beletartozik a LEHET reláció által megadott valamelyik intervallumba, és fordítva, a LEHET reláció minden intervallumába beleesik legalább egy rendelésszám. A következő SQL utasításokban ezeknek a feltételeknek a meglétét feltételezzük.

A ténylegesen hiányzó rendelésszámoknak a meghatározásánál az okozza a gondot, hogy a hiányzó rendelésszámoknak háromféle csoportja lehet:

- hiányozhat egy rendelésszám egy lehetséges rendelésszám intervallum elejéről,
- hiányozhat a végéről és
- hiányozhat az intervallum közepéről.

Ha egy rendelésszám egy lehetséges rendelésszám intervallum elejéről hiányzik, a LEHET táblából kell a hiányzó rendelésszám alsó határát meghatározni, és a felső határt a RENDELÉSSZÁM táblából a következő legkisebb rendelésszám - 1 adja.

Ha egy rendelésszám egy lehetséges rendelésszám intervallum végéről hiányzik, a LEHET táblából kell a hiányzó rendelésszám felső határát meghatározni, és az alsó határt a RENDELÉSSZÁM táblából az előző legnagyobb rendelésszám + 1 adja.

Ha egy rendelésszám egy lehetséges rendelésszám intervallum közepéről hiányzik, az azonos csoporthoz tartozó rendelésszámokból kell a rendelésszámok ugrását meghatározni, vagy az előzőekben meghatározott NEM LÉTEZIK táblának azokat a sorokat kell kiválogatni, amelyeknél a rend számtól és a rend_számgig is ugyanahhoz a LEHET sorhoz tartozik.

A három lekérdezés eredményének az egyesítése adja a ténylegesen hiányzó rendelésszámokat.

Először gyűjtjük ki a LEHET táblából a RENDELÉSSZÁM táblában nem létező kezdőszámokat.

```
SELECT betű &      & Formaitól, '0000') AS rend_szá, betű, tói
FROM lehet
WHERE (betű &      & Formaitól, '0000')) NŐT IN
      (SELECT rend_szá FROM
        rendelésszá);
      -» NEM_ELEJÉN_ALSÓ{rend_szá, betű, tói}
```

A hozzá tartozó felső határ:

```
SELECT nem_elején_alsó.rend_szá AS rend_számtól,
      rendelésszá.betű & '-' & Format(rendésszá.szám-
      1,'C000') AS rend_száig
FROM nem_elején_alsó, rendelésszá
WHERE rendelésszá.rend_szá = (SELECT MIN(rend_szá)
FROM      rendelésszá      WHERE      rend_szá      >
nem_elején_alsó.rend_szá);
      NEM_ELEJÉN{rend számtól, rend_száig}
```

Hasonlóan gyűjtjük ki a LEHET táblából a RENDELÉSSZÁM táblában nem létező befejező számokat.

```
SELECT betű & '-' & Formatíig, '0000') AS rend_szá, betű, ig FROM
      lehet
WHERE betű & '-' & Formaiig, '0000') NŐT IN (SELECT
      rend_szá FROM rendelésszá);
      NEM_VÉGÉN_FELSŐ{rend_szá, betű, ig}
```

A hozzá tartozó alsó határ:

```
SELECT rendelésszá.betű &      &
      Format(rendésszá.szám+1,'0000') AS rend_számtól,
      nem_végén_felső.rend_szá AS rend_száig
FROM nem_végén_felső, rendelésszá
WHERE rendelésszá.rend_szá =
      (SELECT MAX(rend_szá) FROM rendelésszá
      WHERE rend_szá < nem_végén_felső.rend_szá);
      NEM_VÉGÉN{rend_számtól, rend_száig}
```

Egy lehetséges rendelésszám intervallum közepéről hiányzó rendelésszámok meghatározásához a NEM_LÉTEZIK táblából induljunk ki. A NEM_LÉTEZIK egy sora akkor jó, ha benne van a LEHET tábla ugyanabban a sorában, ezért először határozzuk meg, hogy a NEM_LÉTEZIK táblában lévő két rendelésszám a LEHET tábla melyik sorához tartozik.

```
SELECT rend_számtól, első.sorszám AS sorszámtól, rend_számig,
       másod.sorszám AS sorszámig
FROM nemjétezik, lehet AS első, lehet AS másod WHERE
rend_számtól >=
    első.betű &      &Format(első.tól, '0000')
AND rend_számtól <=
    első.betű &      &Format(első.ig, '0000')
AND rend_számig >=
    másod.betű & '-' & Format(másod.tól, '0000')
AND rend_számig <=
    másod.betű & '-' & Format(másod.ig, '0000');
-> NEM_KÖZÉP_SOR{rend_számtól, sorszámtól, rend_számig,
                sorszámig}
```

A NEM_KÖZÉP_SOR tartalmazhat olyan sorokat is, amelyeket már kiválasztottunk, de majd az egyesített lekérdezésben, az UNIÓN miatt, végül minden sor csak egyszer fog megjelenni.

Csak azok az intervallumok kellenek, amelyek ugyanahhoz a LEHET táblában megadott intervallumhoz tartoznak, azaz ahol a két sorszám megegyezik.

```
SELECT rend_számtól, rend_számig
FROM nem_közép_sor WHERE sorszámtól = sorszámig;
+ NEM_KÖZEPÉN{rend_számtól, rend_számig}
```

A három tábla egyesítése:

```
SELECT rend_számtól, rend_számig
FROM nem_elején
UNIÓN
SELECT rend_számtól, rend_számig
FROM nem_közepén
UNIÓN
SELECT rend_számtól, rend_számig
FROM nem_végén
ORDER BY rend_számtól;
NEM_LÉTEZIK_TÉNY{rend_számtól, rend_számig}
```

A nem létező rendelésszámok darabszáma:

```
SELECT SUM(Cint(Right(rend_számg,4)) -  
           Cint(Right(rend_számtól,4)) +1 ) AS darabszám  
FROM nem_létezik_tény;
```

Gyakorló feladatok

1. Tegyük fel, hogy a három számjegyből álló anyagazonosítónak van egy belső szerkezete. Az első számjegy egy anyagcsoport azonosító, a következő két számjegy egy sorszám. Adjuk meg anyagcsoportonként a lehetséges azonosító intervallumokat, majd ellenőrizzük, hogy minden anyagazonosító beletartozik-e valamelyik megadott intervallumba! Készítsünk egy kimutatást anyagcsoportonként arról, hogy a lehetséges anyagazonosítókból melyek a hiányzó anyagazonosító intervallumok!
2. Keressük meg, hogy a mai naptól kezdődően milyen időintervallumokra nem vállaltak el semmilyen termék előállítását (szállítás szempontjából melyek a szabad napok)!
3. Készítsünk egy táblázatot, amelyben egy sorszám és az adott év szombatjainak a dátuma van benne! Az előző feladatban kapott intervallumokat daraboljuk úgy fel, hogy a szombatok és vasárnapok ne szerepeljenek a szállítás szempontjából szabad időintervallumokban!

30. Az árak változásának követése

A termékek egy részének az árát megváltoztatjuk. Vizsgáljuk meg, hogy hogyan hat az árváltozás az adatbázis szerkezetére, mit kell tenni, hogy ne veszítsük el az árváltozás előtti megrendelések idejében érvényben lévő árakat!

A jelenlegi adatbázisban a termékek tulajdonsága a termék ára. Ebből következik, hogy egy megadott terméknek csak egy ára lehet, és ezt a TERMÉK táblában tároltuk. Ha bármelyik megrendelésben lévő termék árát ki szeretnénk számítani, a TERMÉK táblában találjuk a minden időben érvényes, a termékre jellemző árakat. Ha ezt az árat felülírjuk, elveszítjük a megrendelés időpontjában érvényes termékárát, aminek következtében egyetlen régebbi megrendelés összértékét sem tudjuk az akkor érvényben lévő árakon megbízhatóan kiszámítani. Még azt sem tudjuk, hogy a megrendelésben lévő termékeknek megváltozott-e időközben az ára. Ha feltételezzük, hogy az árak változhatnak, az ár, mint tulajdonság, már nemcsak a termék kódjától fog függeni, hanem az időtől is.

Az árváltozás kezelésére a gyakorlatban általában háromféle technikát alkalmazhatunk.

I. Modell

Az egyik megoldás, hogy a TERMÉK tábla mindig a napi aktuális árat tartalmazza, és a megrendeléshez a RENDELÉS táblába eltesszük a termék mellé a rendelés idejében élő aktuális árat is.

	rend szám	
rend szám	kód	kód
RENDELÉSFEJ	RENDELÉS	TERMÉK
partner kód	darab	név
rend dátum	rend ár	napiár
	dátum	
	kész	

A megoldás előnye, hogy az árváltozás kezelése egyszerű. A számlakészítés idején a napi árral tudunk dolgozni, egy adott megrendelésnél az árat pedig a rendelés idején érvényben lévő árral tudjuk kiszámítani, amely ugyanabban a táblázatban és ugyanabban a sorban szerepel, mint maga a rendelés. Hátránya, hogy az adatbázis redundanciát tartalmaz, hiszen egy termék árát annyszor tesszük el, ahányszor a két árváltozás közötti időszakban azt a terméket megrendelték.

További hátránya, hogy csak azok a régi árak tárolódnak az adatbázisban, amelyeken történt értékesítés, továbbá egy árváltozást csak akkor tudunk az adatbázisba eltárolni, amikor az aktuálissá válik.

1.1. Határozzuk meg, hogy mennyi az „AA6” kódú termék mai ára!

```
SELECT kód, napi_ár FROM termék WHERE kód = 'AA6';
```

1.2. Határozzuk meg, hogy mi volt az „AA6” kódú terméknek az ára, amikor értékesítették az „Xy-1234” rendelésszámú megrendelésben!

Jelenítsük meg azt is, hogy mikor történt a megrendelés!

```
SELECT kód, rend_ár, rend_dátum
FROM rendelésfej, rendelés
WHERE rendelésfej.rend_szám = rendelés.rend_szám
AND kód = 'AA6'
AND rendelés.rend_szám = 'Xy-1234';
```

II. Modell

Ha az adatbázisban tárolandó adatokat redundancia mentesen, harmadik normál formában szeretnénk tárolni, egy újabb táblát kell felvennünk, amely az árváltozásokat tartalmazza. Az új árhoz azt az időpontot tároljuk, amikor egy termék ára megváltozott. Egy termék árat egy adott időpontban a termék kódja és az adott időpont előtti legutolsó árváltozáshoz tartozó ár határozza meg. A dátumot, ha egy nap egy terméknek csak egy ára lehetséges, elég nap pontossággal tárolni, különben az árváltozást annak a dátumának és az időpontjának a segítségével tudjuk azonosítani, amikor az ár megváltozott. Ebben az esetben természetesen a RENDELÉSFEJ táblában is ugyanolyan pontossággal kell a rend dátum mezőt kitölteni, mint az ÁRVÁLTOZÁS táblában lévő dátum mezőt.

	rend szám		kód
rend szám	kód	kód	dátum
RENDELÉSFEJ	RENDELÉS	TERMÉK	ÁRVÁLTOZÁS
partnerkod	darab	név	ár
renddátum	dátum		
	kész		

A megoldás előnye, hogy minden árat csak egyszer tárolunk, hátránya azonban, hogy mind a megrendelés idején, mind egy korábbi megrendeléshez tartozó

árak kiszámítása idején az aktuális árat egy nem túl egyszerű algoritmussal tudjuk csak meghatározni. További előnye, hogy egy tervezett árváltozás bármikor bevihető az ÁRVÁLTOZÁS táblába. Figyelembe kell venni azt is, hogy az ÁRVÁLTOZÁS tábla jelentős mennyiségű terméknél és gyakori árváltozásnál igen nagyra is nőhet.

A két megoldás információtartalmában is van különbség. Ha egy adott múltbeli időpontban egy termék árára vagyunk kíváncsiak, az első megoldásból csak akkor tudjuk meghatározni, ha az akkor érvényes árral történt eladás, azaz az első megoldásból nem az adott időponthoz tartozó árat tudjuk meghatározni, hanem az adott időpontnál nem nagyobb, időben hozzá legközelebb eső, eladásnál érvényes árat.

A továbbiakban tegyük fel, hogy az ÁRVÁLTOZÁS tábla nem tartalmaz jövőben bevezetendő árakat, amiből következik, hogy az aktuális ár mindig a legutolsó dátummal eltett ár!

II.1. Határozzuk meg, hogy mennyi az „AA6” kódú termék mai ára!

Az ÁRVÁLTOZÁS táblának az „AA6” termékkódot tartalmazó soraiból azt a sort kell kiválasztani, amelyik a legnagyobb dátumot tartalmazza.

```
SELECT kód, ár
FROM árváltozás
WHERE kód = 'AA6'
      AND dátum = (SELECT MAX(dátum)
                  FROM árváltozás
                  WHERE kód = 'AA6');
```

II.2. Határozzuk meg, hogy mi volt az „AA6” kódú termék ára 2002. január 10-én!

Az ÁRVÁLTOZÁS táblának az „AA6” termékkódot tartalmazó soraiból azt a sort kell kiválasztani, amelyik az adott dátumnál nem nagyobb dátumot tartalmazó sorokból a legnagyobb dátumot tartalmazza. Mivel az ÁRVÁLTOZÁS táblának a kulcsa a kód és a dátum mező együttesen, ha van a kérdésnek megfelelő sor, akkor csak egyetlen ilyen sor lehet. Ha a terméket 2002. január 1.-én még nem forgalmaztuk, akkor a lekérdezés nem ad vissza egyetlen sort sem.

```
SELECT kód, ár
FROM árváltozás
WHERE kód = 'AA6'
      AND dátum =
      (SELECT MAX(dátum)
       FROM árváltozás
       WHERE kód = 'AA6'
       AND dátum <= #01/10/02#);
```

II.3. Készítsük el a 2002. január 1.-én érvényben volt árlistát!

Először a termékkódokhoz határozzuk meg a 2002. január 1-i árakat, majd a termékkódok alapján válogassuk hozzá a termékneveket. Tegyük fel, hogy a TERMÉK táblában minden olyan termék szerepel, amelyet 2002. január 1.-én forgalmaztunk. Ebben a listában az utolsó érvényes árral benne lesz minden olyan termék is, amelyet már nem forgalmazunk. (A termék forgalmazásának befejezésére vonatkozó információ nincs az adatbázisban. Egy valódi adatbázisban természetesen ezt a kérdéskört is pontosan meg kell tervezni! Garanciális okokból a termékre vonatkozó információt akkor sem törölhetjük az adatbázisból, ha már nem forgalmazzuk, de bevezethetünk érvényességi kódot, amely megmutatja, hogy a terméket jelenleg is forgalmazzuk. Ha más információkat akarunk tárolni a jelenleg forgalmazott és a már nem forgalmazott termékekről, akkor külön táblában is tárolhatjuk a kétféle termékeket.)

```
SELECT kód, MAX(dátum) AS dát020101
  FROM árváltozás
 WHERE dátum <= #01/01/02#
 GROUP BY kód;
                                KÓD_DÁT{kód, dát020101}
```

```
SELECT árváltozás.kód, ár AS ár020101
  FROM árváltozás, kód dát
 WHERE árváltozás, kód = kód_dát.kód
       AND dátum = dát020101;
                                KÓD_ÁR{kód, ár020101}
```

```
SELECT név, termék.kód, ár020101
  FROM termék, kód_ár
 WHERE termékkód = kód_ár.kód;
```

A fenti három lekérdezést egyetlen lekérdezésbe is összeépíthetjük.

```
SELECT termék.név, termék.kód, külső.ár
  FROM termék, árváltozás külső WHERE termék.kód =
    külső.kód
    AND külső.dátum =
      (SELECT MAX(belső.dátum) FROM árváltozás belső
       WHERE belső.kód = külső.kód
        AND dátum <= #01/01/02#);
```

11.4. Számítsuk ki, hogy átlagosan hány százalékkal emelkedtek az árak az év elejéhez képest!

Az összehasonlítás alapjául az elmúlt év december 31,-i árakat vegyük figyelembe, és akkor a fentiek alapján az elmúlt év végi árlistát adó lekérdezés:

```
SELECT kód, ár
FROM árváltozás külső
WHERE dátum =
      (SELECT MAX(belső.dátum)
       FROM árváltozás belső
       WHERE belső.kód = külső.kód AND
       Year(belső.dátum) <= Year(DateO) - 1);
      KÓD_ÁR_T{kód, ár}
```

A mai árlista:

```
SELECT kód, ár
FROM árváltozás külső
WHERE dátum =
      (SELECT MAX(belső.dátum) FROM árváltozás
       belső WHERE belső.kód = külső.kód);
      -» KÓD_ÁR_M{kód, ár}
```

Termékenként az árnövekedés átlaga (ebben az átlagba csak azoknak a termékeknek az árai kerülnek bele, amelyeket már az elmúlt évben is forgalmaztunk):

```
SELECT AVG(kód_ár_m.ár / kód_ár_t.ár) - 1 AS növekedés FROM
      kód_ár_m, kód_ár_t WHERE kód_ár_m.kód = kód_ár_t.kód;
```

11.5. Számítsuk ki, hogy a tárgyévi bevétel mennyivel és hány százalékkal lett volna kevesebb, ha év közben nem emeltünk volna árakat!

A változatlan áron számított tárgyévi bevétel kiszámításához használjuk fel az előző pontban előállított KÓD ÁR T árlistát. (Ebben természetesen nincsenek benne azok a termékek, amelyeket tavaly nem forgalmaztunk.)

```
SELECT SUM(darab * kód_ár_t.ár) AS v_bevétel FROM rendelés,
      kód_ár_t WHERE rendelés.kód = kód_ár_t.kód AND Yea^'áuím) =
      Year(Date());
      BEVÉTEL_V{v_bevétel}
```

A tárgyévi tényleges bevétel kiszámításához a megrendelés időpontjában érvényes árakat kell figyelembe venni. Az összehasonlíthatóság érdekében csak azokat a megrendeléseket vehetjük figyelembe, amelyeket az elmúlt évben is forgalmaztunk, azaz amelyeknek létezik tavalyi ára is.

```
SELECT SUM(darab * külső.ár) AS tény_bebevétel
FROM rendelés, árváltozás külső WHERE rendelés.kód =
külső.kód
AND Year(rendelés.dátum) = Year(Date())
AND külső.dátum =
(SELECT MAX(belső.dátum)
FROM árváltozás belső
WHERE belső.kód = külső.kód
AND belső.dátum <= rendelés.dátum)
AND EXISTS
(SELECT tavalyi.dátum
FROM árváltozás tavalyi
WHERE tavalyi.kód = külső.kód AND
Year(tavalyi.dátum) <= Year(Date()) -
1);
BEVÉTEL_TÉNY{tény_bebevétel}
```

Mind a BEVÉTEL V, mind a BEVÉTEL_TÉNY egy soros és egy oszlopos tábla, ezért a bevételek közötti különbség:

```
SELECT v_bebevétel
AS [változatlan áron számított tárgyévi bevétel],
tény_bebevétel AS [tárgyévi tényleges bevétel],
tény_bebevétel - v_bebevétel AS különbség, (1 - v_bebevétel
/ tény_bebevétel) * 100
AS [% lett volna a veszteség]
FROM bevétel_v, bevétel_tény;
```

III. Modell

A két megoldást kombináltan is használhatjuk. A TERMÉK táblában az aktuális napi árat, és az ÁRVÁLTOZÁS táblában a régi árakat tároljuk. Az árváltozás dátuma azt fogja tartalmazni, hogy az adott ár meddig volt érvényben, beleértve még az adott napot is. Ezzel a módszerrel a megrendelés idején gyorsan tudjuk meghatározni a fizetendő árat, és az árváltozás táblához viszonylag ritkábban, pl. zárások idején vagy számlamásolat idején, kell csak fordulni. Egyik esetben sem kritikus a feldolgozási idő. Amikor a megrendeléseket archiváljuk, a megrendeléssel együtt az ÁRVÁLTOZÁS táblát is eltesszük, utána kiürítjük az ÁRVÁLTOZÁS táblát.

Ha szükség van rá, a rendszerben egy jövőbeli árváltozásra vonatkozó információt is tárolhatunk, ami elemzésekhez nagyon hasznos lehet.

Az előző modellhez képest a legszembetűnőbb előnye ennek a modellnek, hogy a nyitóárakat nem kell felvenni az ÁRVÁLTOZÁS táblába, ami nagy TERMÉK táblák esetén igen hasznos lehet.

Ha minden árat, a jelenlegi és a régi árakat is egy adott feladatnál egyszerre akarunk kezelni, egy táblaegyesítő lekérdezéssel ideiglenesen előállíthatunk egy az előző modellhez hasonló árváltozás táblát, de itt a dátum azt fogja jelenteni, hogy meddig volt érvényben az adott ár.

	rend szám		kód
rend szám	kód	kód	dátumig
RENDELÉSFEJ	RENDELÉS	TERMÉK	ÁRVÁLTOZÁS
partner kód	darab	név	régiár
rend dátum	dátum	akt ár	
	kész		

111.1. Határozzuk meg, hogy mennyi az „AA6” kódú termék mai ára!

```
SELECT kód, akt_ár FROM termék WHERE kód = 'AA6';
```

111.2. Határozzuk meg, hogy mi volt az „AA6” kódú termék ára 2002. január 10-én!

Nem tudhatjuk, hogy melyik táblában van az „AA6” kódú terméknek az adott dátum idejében érvényes ára. Ha utoljára 2002. január 10. előtt emelték a terméknek az árát, akkor a kívánt ár a TERMÉK táblában van, hiszen ma is ez az aktuális ár. Ellenkező esetben az ár az ÁRVÁLTOZÁS táblában van.

Készítsünk egy lekérdezést, amelyik az „AA6” kódú termék minden árát tartalmazza. Az ÁRVÁLTOZÁS táblából hagyjuk el azt a bejegyzést, amely a jövőre vonatkozik, ha ilyen van:

```
SELECT kód, régi_ár AS ár, dátumig
  FROM árváltozás
 WHERE kód = 'AA6'
    AND dátumig IS NOT NULL
UNION
SELECT kód, akt_ár, Date()
  FROM termék;
```

■> AA6ÁR{kód, ár, dátumig}

Ebből az eredménytáblából már bármilyen dátumhoz tartozó árat meghatározhatunk.

```
SELECT kód, ár
FROM aa6ár
WHERE dátumig =
      (SELECT MIN(dátumig)
       FROM aa6ár
       WHERE dátumig >= #01/10/02#);
```

Gyakorló feladatok

1. Számítsuk ki a I. modell szerint, hogy mennyi volt a bevétel az elmúlt évben, és mennyi lett volna a tavalyi bevétel a jelenlegi árakon!
2. Számítsuk ki a III. modell szerint, hogy mennyi volt a bevétel az elmúlt évben, és mennyi lett volna a tavalyi bevétel a jelenlegi árakon!
3. AIII. modell szerint készítsük el a 2002. január 1.-én érvényben volt árlistát!

Gyakorló feladatok

A következő feladatcsoportot önálló gyakorlásra szánjuk. A feladatok sorrendje nem jelent nehézségi sorrendet, inkább témák szerint csoportosítottunk.

Három fogalom, amit a feladatok egy részénél fel fogunk használni:

- *veszteséges*: a beépített anyagok összértéke nagyobb, mint a termék árának $\text{IO}\%$ -a,
- *termékcsoport*: a termékeket csoportokba sorolhatjuk a termékkód első karaktere szerint,
- *bonyolultság*: minden termékhez hozzárendelünk egy bonyolultsági mutatót, amit az előállításához szükséges anyagfajták számával adunk meg.

Anyag

1. Melyik anyagot nem használjuk fel egyetlen termékben sem?
2. Az összes termék hány %-ában van benne a „211” azonosítójú anyag?
3. Készítsen egy listát, amelyik minden anyagról megmondja, hogy hány termékben van benne!
4. Melyek azok az anyagok, amelyek a termékeknek több mint 25%-ában benne vannak?
5. Melyik anyagból fogyott az elmúlt hónapban a legtöbb?

Termék

6. Írassuk ki azokat a megrendelőket, akik csak „AA1” kódú terméket vásároltak?
7. Írassuk ki azokat a megrendelőket, akik legalább 5 darab „AA1” kódú terméket vásároltak 2002. januárjában?
8. Írassuk ki azokat a megrendelőket, akik az „AA1” kódú termékből többet vásároltak, mint bármilyen más termékből?
9. Melyek azok a termékek, amelyeket az elmúlt évben nem rendeltek meg?
10. Melyek azok a termékek, amelyek forgalma az elmúlt egy évben a felére esett vissza?

Veszteség

11. Melyek azok a termékek, amelyek veszteségesek?
12. Melyek azok a termékek, amelyekbe beépített anyag összértéke nagyobb, mint a termék árának 150%-a?
13. Melyik terméken van a legnagyobb nyereség?
14. Melyek azok a termékcsoportok, amelyekben nincs veszteséges termék?

Gyakorló feladatok

15. Készítsen három kategóriát: veszteséges, nyereséges és kiemelten nyereséges termék aszerint, hogy a beépített anyagok összértéke a termék árának 70 százalékánál több, 50 és 70 százalék között van, illetve 50 százaléknál kevesebb. Termékcsoportonként melyik kategóriába hány termék tartozik?

Megrendelés egy számlán

16. Havonta mennyi volt az egy rendelésszámon megrendelt termékek legnagyobb összértéke?
17. Mennyi az egy rendelésszámon leadott megrendelések átlagos értéke?
18. Melyek azok a rendelésszámok, ahol a megrendelés értéke nagyobb, mint az átlag megrendelés értéke?
19. Ki volt az elmúlt hónapban a tíz legjobb megrendelő (az a tíz, aki a legnagyobb értékben rendelt meg az elmúlt hónapban)?
20. Ki az a megrendelő, aki egy nap több rendelésszámon is adott megrendelést?

Bevétel

21. Mennyi volt a bevétel 2002. márciusban?
22. Melyik hónapban volt 2001.-ben a legnagyobb a bevétel?
23. 2000-2001. évben melyek voltak azok a hónapok, amikor nem rendeltek meg semmit?
24. Rendezze és sorszámozza is be a 2001. hónapjait aszerint, hogy mennyi volt a megrendelés az „A” betűvel kezdődő termékekből!
25. Határozza meg azokat az időintervallumokat, amikor a megrendelések napi összértéke kisebb volt az átlagos napi megrendelések összértékénél!

Bonyolultság

26. Számítsa ki a termékek átlagos bonyolultságát!
27. Melyek azok a termékek, amelyek az átlagos bonyolultsági szinttől 10%-kal térnek el?
28. Melyek azok a megrendelések, amelyek minden megrendelt terméke az átlagos bonyolultsági szint felett volt?
29. Számítsa ki az egyes termékcsoportok átlagos bonyolultságát!
30. Melyek azok a termékcsoportok, amelyeknek minden terméke az átlagos termékbonyolultsági szint felett van 50 százalékkal?

ACCESS függvények

Az SQL nyelvnek csak adatbázis-kezelő utasításai vannak, nincsenek szabványos mezőkezelő függvényei, mindig azokat a függvényeket használja fel, amelyik környezetbe beépítettük az SQL utasításokat. A példatárban az SQL-t Microsoft ACCESS adatbázis-kezelő környezetben mutattuk be. Ebben a függelékben összefoglaljuk azokat a mezőkezelő függvényeket, amelyeket a példatárban felhasználtuk. Természetesen az itt bemutatottaknál lényegesen több függvényt használhatunk minden olyan környezetben, ahol SQL utasításokat írhatunk. A magyar nyelvű ACCESS-ben a függvénykészlet hasonlít a Microsoft Excel függvényeihez, de nem fordították le a függvényneveket magyarra. További részletek, pontos meghatározások megtalálhatóak az ACCESS súgójában.

A példatárban felhasznált ACCESS függvények:

Asc(karaktorsorozat)

A karaktorsorozat első karakterének megfelelő karakterkódot adja vissza, mint egészszámot eredményként.

Chr(karakterkód)

Az ASCII karakterkód értékének megfelelő karaktert, mint egykarakteres szöveget adja vissza eredményként.

A karakterkód értéke 0 és 255 közötti egészszám lehet.

Cint (kifejezés)

A kifejezés értékét Integer típusúra alakítja.

A kifejezés argumentum tetszőleges érvényes numerikus vagy karakteres kifejezés.

Date()

A számítógép rendszerórájában található dátumot adja vissza. A zárójelpár megadása kötelező.

DateAdd(dátumrész, szám, dátum)

Visszaad egy dátumot, amelyben a megadott számmal növekszik, vagy csökken a dátum meghatározott része.

A dátumrész argumentum a dátum annak a részének a karakteres jelölése, amely a függvény által visszaadott dátumban megváltozik.

Pl.: DateAdd("yyyy",-1,Date()) a rendszeróra szerinti egy évvel ezelőtti dátum.

dátumrész leírás

yyyy	év
m	hónap
d	nap

A dátumrész argumentum további lehetséges értékei az ACCESS súgójában találhatók.

Formát (kifejezés, formátum)

A függvény egy karakterláncot ad vissza, amelyet a kifejezésből állít elő a formátumban meghatározott formázással. Pl.: Format(Year(Date()),"0000") a rendszeróra évét adja vissza, mint négy számjegyből álló karakterláncot.

kifejezés formátum	formázott karakterlánc,
numerikus 0	egészszám vagy 0,
numerikus 0000	egészszám, vezető nullákkal kiegészítve, hogy legalább négy karakteres

További lehetséges argumentumok és értékeik az ACCESS súgójában találhatók.

IIf(kifejezés, igazrész, hamisrész)

A függvény az igazrész vagy a hamisrész értékét adja vissza, a kifejezés értékétől függően. kifejezés kiértékelése után logikai értéket adó kifejezés, igazrész a függvény által visszaadott érték, ha a kifejezés értéke igaz, hamisrész a függvény által visszaadott érték, ha a kifejezés értéke hamis.

Int(szám)

A szám egészrészét adja vissza.

Az Int és Fix függvény egyaránt a szám törtrészét távolítja el, és az eredményül kapott egész számot adja vissza. A különbség az Int és Fix között a negatív szám kezelésében van. Az Int az első negatív egészszámot adja vissza, amely kisebb vagy egyenlő a szám értékével, míg a Fix azt az első negatív számot adja vissza, amely nagyobb vagy egyenlő a szám értékével. Pl.: Int (-8,4) = -9, míg Fix(8,4) = -8.

Left(karakterlánc, hossz)

Az adott karakterlánc bal széléről a hossz által megadott számú karaktert adja vissza.

Month(dátum)

A dátumból az év hónapját adja meg, 1 és 12 közötti egész számmal.

Right(karakterlánc, hossz)

Az adott karakterlánc jobb széléről a hossz által megadott számú karaktert adja vissza.

Rnd(szám)

A függvény egy véletlen számértéket ad vissza, amelynek az értéke nagyobb vagy egyenlő, mint nulla, és kisebb, mint egy.

szám	visszaadott érték
0 hiányzik	az utoljára visszaadott értéket ismétli, a következő véletlen számértéket adja.

További magyarázatot az ACCESS súgójában lehet megtalálni.

Round(szám, decimális hely)

A függvény a szám argumentumot kerekíti a decimális helynek megfelelő tizedes helyre.

Space(szám)

Adott számú szóközből álló karakterláncot ad vissza.

Year(dátum)

A dátumból az évszámot adja meg, egész számként.