Az SQL nyelv

SQL (Structured Query Language = Strukturált Lekérdező Nyelv).

A lekérdezési funkciók mellett a nyelv több olyan elemmel is rendelkezik, amelyek más adatkezelési funkciók végrehajtására is alkalmasak. A nyelv legújabb szabványos változatai pedig már egészen kiterjedt adatbázis-kezelési műveletek megvalósítására is használhatók.

Az SQL szabványos adatbázis-kezelő nyelvvé vált.

Általában a különböző ABKR implementációk a saját képükre formált SQL megvalósítással rendelkeznek.

Van azonban a nyelvnek két fő szabványos változata. Ezek a ANSI (American National Standards Institute = Amerikai Nemzeti Szabványügyi Intézet) által definiált SQL,

illetve az 1992-ben elfogadott módosított szabvány, az

SQL-92 vagy SQL2.

Később ezeket az ISO (International Standars Organization =

Nemzetközi Szabványügyi Szervezet) is elfogadta.

Lekérdezések az SQL-ben

Egyszerű lekérdezések

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

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

A parancs első részében a * jel azt jelenti, hogy a lekérdezés a teljes táblára vonatkozik.

Az <oszlopnévlista> segítségével adhatjuk meg azt, hogy a lekérdezés eredményeképpen melyik oszlopokat kívánjuk megjeleníteni.

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

Amennyiben az oszlopkifejezésben aggregáló függvényt használunk, akkor az eredménytáblában nem a tábla egyes előfordulásainak adatai jelennek meg, hanem azoknak a megfelelő aggregáltját fogja képezni a parancs. A tábla soraiból álló halmazra végrehajtódik a megfelelő függvény.

COUNT: Megadja a tábla sorainak számát.

SUM: Megadja a paraméterében szereplő oszlop adatainak az összegét az összes rekordra. Csak numerikus attribútumra alkalmazható.

AVG: Megadja a paraméterében szereplő oszlop adatainak az átlagát az összes rekordra. Csak numerikus attribútumra alkalmazható.

MIN: Megadja a paraméterében szereplő oszlop adatainak a minimumát az összes rekordra. Csak numerikus attribútumra alkalmazható.

MAX: Megadja a paraméterében szereplő oszlop adatainak a maximumát az összes rekordra. Csak numerikus attribútumra alkalmazható.

Lehetőség van oszlopok átnevezésére is. Ennek szintaxisa a következő:

<oszlopkifejezés> AS <oszlopnév>

Az eredménytáblában az **<oszlopnév>** paraméterben megadott oszlopnév szerepel.

Előfordulhat, hogy egy lekérdezés végrehajtása után a keletkezett elemek között ugyanaz a rekord többször előfordul.

ALL opció alkalmazásakor, illetve alapértelmezés szerint az azonos előfordulások többszörösen fognak megjelenni az eredményben.

Ha **DISTINCT** opciót alkalmazzuk, akkor minden azonos előfordulás csak egyszer jelenik meg az eredménytáblában. A következőkben nézzünk néhány példát az egyszerű lekérdező parancsok alkalmazására:

Példa: **Dolgozó** tábla:

A dolgozó törzsszáma	A dolgozó Neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T234578	Kiss István	Eger	1968. 12. 11.	120000
T456734	Nagy József	Budapest	1972. 01. 30.	150000
T429877	Kovács János	Szeged	1967. 05. 12.	120000

A **teljes tábla** lekérdezése a következő paranccsal történhet:

SELECT * FROM Dolgozó

Példa

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

SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó

A keletkezett eredménytábla a következőképpen néz ki:

A dolgozó neve	A dolgozófizetése
Kiss István	120000
Nagy József	150000
Kovács János	120000

Példa

Készítsünk olyan eredménytáblát, amely tartalmazza a dolgozók nevét, fizetését, valamint a 10%-kal megemelt fizetéseket.

Az új oszlop neve legye **A dolgozó emelt fizetése**. A feladatot az alábbi paranccsal oldhatjuk meg:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése', 1.1*'A dolgozó fizetése' AS
'A dolgozó emelt fizetése' FROM Dolgozó
```

A keletkezett eredménytábla a következő:

A dolgozó neve	A dolgozó fizetése	A dolgozó emelt fizetése
Kiss István	120000	132000
Nagy József	150000	165000
Kovács János	120000	132000

Példa

Jelenítsük meg a dolgozók fizetését úgy, hogy a listában ne legyen két azonos érték. A feladatot az alábbi paranccsal oldhatjuk meg:

```
SELECT DISTINCT 'A dolgozó fizetése' FROM Dolgozó
```

A keletkezett eredménytábla a következő:

A dolgozó fizetése
120000
150000

Példa

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

Az oszlopok nevei legyenek rendre: **A dolgozók száma, A dolgozók összes fizetése, A dolgozók átlagos fizetése**.

A feladatot a következő módon oldhatjuk meg:

```
SELECT COUNT('A dolgozó neve') AS 'A dolgozók száma' , SUM('A dolgozó fizetése') AS 'A dolgozók összes fizetése', AVG('A dolgozó fizetése') AS 'A dolgozók átlagos fizetése' FROM Dolgozó
```

A keletkezett eredménytábla a következő:

A dolgozók száma	A dolgozók összes fizetése	A dolgozók átlagos fizetése
3	390000	130000

Kiválasztó lekérdezések

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

[WHERE <feltétel>]

A feltételben operanduszok és operátorok szerepelhetnek.

Az operátorok összehasonlító (<,>,<=,>=,<>), aritmetikai (+,-,*,/) és

logikai műveletek (AND, OR, NOT), míg az operanduszok lehetnek **konstansok**, **reláció attribútumok**, azaz oszlopnevek, illetve **függvényhivatkozások**.

Létezik néhány **predikátum függvény**

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

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

Az <alsóérték>, illetve <felsőérték> valamely ismert elemi típusnak (numerikus, dátum) megfelelő konstans,

a <oszlopkifejezés> ugyanilyen típusú kifejezés, amely oszlopnevekből van képezve.

Eredményként azok a rekordok fognak eleget tenni a feltételnek, amelyekre a kifejezés értéke a két konstans közé esik.

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

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

A kifejezés hatására annak vizsgálata történik meg, hogy az <oszlopkifejezés> értéke szerepel-e a megadott értéklistában, vagy nem.

Amennyiben szerepel a kifejezés értéke igaz lesz. Amennyiben használjuk a **NOT** kulcsszót, a kifejezés akkor lesz igaz, ha az értéke nem szerepel a listában. Az <**értéklista**> paraméterben tehát a kifejezés típusának megfelelő értékeket kell vesszővel elválasztva felsorolni.

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

[<oszlopkifejezés> LIKE <karakterlánc>]

A <karakterlánc> konstansban idézőjelek között adhatunk meg karaktersorozatot.

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

Az <oszlopkifejezés> paraméternek karakteres értéket kell szolgáltatni, amely összehasonlításra kerül a konstanssal. Ha a konstansban a **%** jelet használjuk, akkor a két karakterláncnak csak eddig a jelig kell egyezni ahhoz, hogy a feltétel igaz legyen. Az **_** jelet többször is alkalmazhatjuk,

ilyenkor az összehasonlításnál a megadott pozíción bármilyen jel szerepel, azon a helyen az egyezésnek fog számítani.

Példa

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

A feladatot a következő módon oldhatjuk meg:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó WHERE 'A Dolgozó fizetése' >= 150000
```

A keletkezett eredménytábla a következő:

A dolgozó neve	A dolgozó fizetése
Nagy József	150000

Példa

Készítsünk olyan táblázatot, amely megadja a vállalat azon dolgozóinak nevét és fizetését, akik 1960.01.01-e után születtek, és a fizetésük 100000 Ft felett van!

A feladatot a következő módon oldhatjuk meg:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó WHERE 'A dolgozó születési ideje' > {1960.01.01.} AND 'A dolgozó fizetése' >= 100000
```

A keletkezett eredménytábla a következő:

A dolgozó neve	A dolgozó fizetése
Kiss István	120000
Nagy József	150000
Kovács János	120000

Példa

Készítsünk olyan táblázatot, amely megadja a vállalat azon dolgozóinak nevét és fizetését, akiknek a fizetése 100000 és 120000 Ft közé esik!

A feladatot a következő módon oldhatjuk meg:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó WHERE 'A dolgozó fizetése' BETWEEN 100000 AND 120000
```

A feladatot megoldhatjuk másképpen is:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó WHERE 'A dolgozó fizetése' >= 100000 AND 'A dolgozó fizetése' <= 120000
```

A keletkezett eredménytábla a következő:

A dolgozó neve	A dolgozó fizetése
Kiss István	120000
Kovács János	120000

Példa

Készítsünk olyan táblázatot, amely megadja a vállalat azon dolgozóinak nevét és születési helyét, akik Egerben, Szegeden vagy Debrecenben születettek!

A feladatot a következő módon oldhatjuk meg:

```
SELECT 'A dolgozó neve', 'A dolgozó születési helye' FROM Dolgozó
WHERE 'A dolgozó születési helye' IN ("Eger", "Szeged", "Debrecen")
```

A keletkezett eredménytábla a következő:

A dolgozó neve	A dolgozó születési helye
Kiss István	Eger
Kovács János	Szeged

Példa

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

A feladatot a következő módon oldhatjuk meg:

```
SELECT * FROM Dolgozó WHERE 'A dolgozó neve' LIKE "Kovács%"
```

A keletkezett eredménytábla a következő:

A dolgozó törzsszáma	A dolgozó neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T429877	Kovács János	Szeged	1967. 05. 12.	120000

Példa

Készítsünk olyan táblázatot, amely megadja a vállalat T4-gyel kezdődő törzsszámú nem szegedi vagy debreceni születésű, vagy T2-vel kezdődő törzsszámú 1968-ban született dolgozóit!

Tegyük fel, hogy a rendszerünkben egy dátum típusú adat évének lekérdezésére a **YEAR(<Dátumkifejezés>)** függvény használható.

A feladatot a következő módon oldhatjuk meg:

```
SELECT * FROM Dolgozó WHERE ((LEFT('A dolgozó törzsszáma',2) LIKE "T4") AND ('A dolgozó születési helye' NOT IN ("Szeged", "Debrecen"))) OR ((LEFT('A dolgozó törzsszáma',2) LIKE "T2") AND (YEAR('A dolgozó születési ideje') = 1968))
```

A keletkezett eredménytábla a következő:

A dolgozó törzsszáma	A dolgozó neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T234578	Kiss István	Eger	1968. 12. 11.	120000
T456734	Nagy József	Budapest	1972. 01. 30.	150000

Csoportosító lekérdezések és rendezések

A csoportosítás azt jelenti, hogy a rekordokat egy adott mező értékei szerint csoportokra bontjuk.

Ezután a csoportokhoz tartozó rekordokra különböző műveleteket hajthatunk végre, például alkalmazhatjuk a már ismert aggregációs függvényeket, esetleg a csoportokra vonatkozóan kiválasztó műveletet alkalmazhatunk.

A csoportosítás a következő utasítással hajtható végre:

GROUP BY <oszlopnév>,[<oszlopnév>]…

A csoportosítás a megadott oszlopnevek azonos értékei alapján fog történni. A művelet végrehajtása után minden egyes csoportra egy sor keletkezik az eredménytáblában.

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

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

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

Az alparancs formája az alábbi:

HAVING <csoportfeltétel>

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

Rendezés. Az SQL lehetőséget biztosít arra, hogy lekérdezéseink eredményét rendezetten jelenítsük meg.

ORDER BY alparancs. Formája a következő:

ORDER BY <oszlopnév|oszlopsorszám> [ASC|DESC], <oszlopnév|oszlopsorszám> [ASC|DESC]]…

Egyrészt hagyományosan a nevükkel, másrészt egy számmal, ami a táblázatban az oszlop sorszáma. A számozás 1-től kezdődik a táblázat fejrészében megadott sorrend szerint.

Tegyük fel ezúttal, hogy a **Dolgozó** táblánk az alábbi módon néz ki:

A dolgozó törzsszáma	A dolgozó neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T234578	Kiss István	Eger	1968. 12. 11.	120000
T3443234	Kiss Timót	Eger	1970. 02. 28.	105000
T456734	Nagy József	Budapest	1972. 01. 30.	150000
T768545	Vári Ödön	Budapest	1958. 07. 12.	210000
T429877	Kovács János	Szeged	1967. 05. 12.	120000

Készítsünk olyan listát, amely megadja városonként, hogy az adott városban hány dolgozó született!

```
SELECT 'A dolgozó születési helye' AS 'Születési hely', COUNT('A dolgozó törzsszáma') AS 'A dolgozók száma' FROM Dolgozó GROUP BY 'A dolgozó születési helye'
```

Eredményképpen a következő táblát kapjuk:

Születési hely	A dolgozók száma
Eger	2
Budapest	2
Szeged	1

Példa

Készítsünk olyan listát, amely megadja városonként, hogy az adott városban született dolgozóknak mennyi az összes illetve az átlagfizetése!

```
SELECT 'A dolgozó születési helye' AS 'Születési hely', SUM('A dolgozó fizetése') AS 'A dolgozók összes fizetése', AVG('A dolgozó fizetése') AS 'A dolgozók átlagfizetése' FROM Dolgozó GROUP BY 'A dolgozó születési helye'
```

Eredményképpen a következő táblát kapjuk:

Születési hely	A dolgozók összes fizetése	A dolgozók átlagfizetése
Eger	225000	112500
Budapest	360000	180000
Szeged	120000	120000

Példa

Készítsünk olyan listát, amely megadja azokat a városokat, amelyekre igaz, hogy az adott városban született dolgozóknak az átlagfizetése legfeljebb 120000 Ft!

```
SELECT 'A dolgozó születési helye' AS 'Születési hely', AVG('A dolgozó fizetése') AS 'A dolgozók átlagfizetése' FROM Dolgozó GROUP BY 'A dolgozó születési helye' HAVING AVG('A dolgozó fizetése') < 120000
```

Példa

Készítsünk olyan listát, amely a dolgozók nevét és fizetését név szerint ábécé sorrendben adja meg!

Az alábbi parancsot használhatjuk:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó ORDER BY 'A dolgozó neve'
```

A keletkezett eredménytábla az alábbi:

A dolgozó neve	A dolgozó fizetése
Kiss István	120000
Kiss Timót	105000
Kovács János	120000
Nagy József	150000
Vári Ödön	210000

Példa

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

Az alábbi parancsot használhatjuk:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó ORDER BY 'A dolgozó fizetése' DESC, 'A dolgozó neve'
```

A keletkezett eredménytábla az alábbi:

A dolgozó neve	A dolgozó fizetése
Vári Ödön	210000
Nagy József	150000
Kiss István	120000
Kovács János	120000
Kiss Timót	105000

Több táblára vonatkozó lekérdezések

A legtöbb esetben az adatbázis szerkezete olyan, hogy a szükséges információk több táblában találhatók. Különösen igaz ez, ha normalizált relációkkal dolgozunk, hiszen mint láttuk, a normalizálás alaptevékenysége a több relációra bontás. Ilyen esetekben az információk összegyűjtéséhez minden táblára szükségünk van, ezért a lekérdezéseket ki kell terjeszteni úgynevezett **többtáblás lekérdezésekké**.

Ennek általános formája a következő:

SELECT [ALL|DISTINCT] * | <oszlopnévlista> FROM <táblanév> [<másodnév>][,<táblanév> [<másodnév>]]…

A kérdés az, hogyan kell értelmeznünk azt, amikor a FROM kulcsszó után több tábla neve szerepel. A választ a relációs modell adja.

Az SQL nyelv lehetőséget biztosít az összekapcsolás relációalgebrai művelet közvetlen megvalósítására is. Ez azt jelenti, hogy speciális utasítások állnak rendelkezésre, amelyek az összekapcsolás különböző fajtáit adják meg.

A legelső ilyen parancs magát a **Descartes szorzatot** hozza létre. A parancs megadásánál csak a FROM kulcsszó utáni részt definiáljuk:

<táblanév> CROSS JOIN <táblanév>

A parancs hatására a megadott két tábla Descartes szorzatát képezi a rendszer.

Mint tudjuk, sokkal természetesebb a **feltételen alapuló összekapcsolás**.

Ennek formája az alábbi:

<táblanév> JOIN <táblanév> ON <feltétel>

Az összekapcsolások egy speciális fajtáját jelentik az úgynevezett

külső összekapcsolások.

Ezek tulajdonképpen annak a problémának a kezelésére használhatók, ami akkor jelentkezik, ha a két összekapcsolandó tábla valamely sorához nem tartozik a másik táblából elem. Ilyenkor az összekapcsolás művelet definíciója alapján ez a sor nem kerül be az eredménytáblába.

Egy külső összekapcsolás abban különbözik a hagyományostól, hogy az eredménybe minden olyan sor is bekerül, amely a másik tábla egyetlen sorához sem kapcsolódik. Egy külső összekapcsolás háromféle módon valósulhat meg.

Az egyik mód az, amikor **mindkét tábla** "lógó" sorai bekerülnek az eredménybe, a másik kettő pedig amikor csak a **bal**, illetve a **jobboldali** tábláé.

Ennek megfelelően a következő esetek lehetnek:

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

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

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

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

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

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

Tekintsük most a Dolgozó és Kifizetés táblákat, amelyek az alábbi módon néznek ki:

A dolgozó törzsszáma	**A dolgozó neve**	**A dolgozó születési helye**	**A dolgozó születési ideje**	**A dolgozó fizetése**
T234578	Kiss István	Eger	1968. 12. 11.	120000
T343234	Kiss Timót	Eger	1970. 02. 28.	105000
T456734	Nagy József	Budapest	1972. 01. 30.	150000
T768545	Vári Ödön	Budapest	1958. 07. 12.	210000
T429877	Kovács János	Szeged	1967. 05. 12.	120000

A kifizetés dátuma	A kifizetett bér	A levont adóelőleg	A dolgozó törzsszáma
2000.01.02.	76000	45000	T234578
2000.01.02	69000	43000	T343234
2000.01.02.	90000	50000	T456734

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

Az alábbi parancsot használhatjuk:

```
SELECT 'A dolgozó neve', 'A kifizetett bér', 'A levont adóelőleg', 'A kifizetés dátuma' FROM Dolgozó, Kifizetés WHERE Dolgozó.'A dolgozó törzsszáma' = Kifizetés.'A dolgozó törzsszáma' ORDER BY 'A dolgozó neve'
```

Ugyanez megvalósítható más módon is:

```
SELECT 'A dolgozó neve', 'A kifizetett bér', 'A levont adóelőleg', 'A kifizetés dátuma' FROM Dolgozó JOIN Kifizetés ON Dolgozó.'A dolgozó törzsszáma' = Kifizetés.'A dolgozó törzsszáma' ORDER BY 'A dolgozó neve'
```

A keletkezett eredménytábla az alábbi:

A dolgozó neve	A kifizetett bér	A levont adóelőleg	A kifizetés dátuma
Kiss István	76000	45000	2000.01.02.
Kiss Timót	69000	43000	2000.01.02
Nagy József	90000	50000	2000.01.02.

Példa

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

A megoldás a következő:

```
SELECT 'A dolgozó neve', 'A kifizetett bér', 'A levont adóelőleg', 'A kifizetés dátuma' FROM Dolgozó LEFT OUTER JOIN Kifizetés ON Dolgozó.'A dolgozó törzsszáma' = Kifizetés.'A dolgozó törzsszáma' ORDER BY 'A dolgozó neve'
```

A keletkezett eredménytábla az alábbi:

A dolgozó neve	A kifizetett bér	A levont adóelőleg	A kifizetés dátuma
Kiss István	76000	45000	2000.01.02.
Kiss Timót	69000	43000	2000.01.02
Kovács János			
Nagy József	90000	50000	2000.01.02.
Vári Ödön			

Példa

Készítsünk olyan listát, amely az azonos városban született dolgozókat listázza ki páronként, úgy hogy a listán egy pár csak egyszer szerepeljen!

Az alábbi parancsot használhatjuk:

```
SELECT Dolgozó1.'A dolgozó neve', Dolgozó2.'A dolgozó neve' FROM Dolgozó Dolgozó1, Dolgozó Dolgozó2

WHERE Dolgozó1.'A dolgozó neve' <> Dolgozó2.'A dolgozó neve' AND Dolgozó1.'A dolgozó neve' < Dolgozó2.'A dolgozó neve' AND Dolgozó1.'A dolgozó születési helye' = Dolgozó2.'A dolgozó születési helye'
```

Figyeljük meg a másodnevek használatát, valamint azt, hogy a feltételek között a Descartes szorzatból kiszűrjük az egyes rekordoknak az önmagukkal való szorzatát, továbbá a

Dolgozó1.'A dolgozó neve' < Dolgozó2.'A dolgozó neve'

feltétellel azt, hogy egy pár kétszer szerepeljen a listában.

A keletkezett eredménytábla az alábbi:

Dolgozó1.A dolgozó neve	Dolgozó2.A dolgozó neve
Kiss István	Kiss Timót
Nagy József	Vári Ödön

Beágyazott lekérdezések

Az SQL nyelv lehetőséget biztosít arra, hogy a kiválasztó lekérdezések feltételében is használjunk SQL lekérdező parancsot. Ilyenkor megkülönböztetünk külső és belső SELECT parancsot, és az ilyen jellegű lekérdezéseket

beágyazott lekérdezéseknek nevezzük.

A SELECT parancsok szintaxisa megegyezik a már ismert formákkal, csupán arra kell ügyelnünk, hogy a belső lekérdezésekre vonatkozóan bizonyos megkötéseknek kell teljesülnie. A belső lekérdezés jellege alapján több esetet különböztethetünk meg.

Ezek a következők:

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

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

Az alábbi három predikátum létezik:

<oszlopkifejezés> [NOT] IN <belső lekérdezés>

Ennél a típusnál az <oszlopkifejezés> paraméterben megadott kifejezés értékéről fogja eldönteni a rendszer, hogy szerepel-e a belső lekérdezés által előállított oszlop adatai között. Ha igen a feltétel értéke igaz, ha nem, akkor hamis lesz.

A következő predikátumok formája az alábbi:

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

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

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

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

[NOT] EXISTS <belső lekérdezés>

Példa

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

Az alábbi parancsot használhatjuk:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó WHERE 'A dolgozó fizetése' < (SELECT AVG('A dolgozó fizetése') FROM Dolgozó)
```

A keletkezett eredménytábla az alábbi:

A dolgozó neve	A dolgozó fizetése
Kiss István	120000
Kiss Timót	105000
Kovács János	120000

Példa

Készítsünk olyan listát, amely a Dolgozó táblából kilistázza azon dolgozók nevét és fizetését, akik a fizetése a legnagyobb fizetéstől legfeljebb csak 60000 Ft-tal tér el!

Az alábbi parancsot használhatjuk:

```
SELECT 'A dolgozó neve', 'A dolgozó fizetése' FROM Dolgozó WHERE 'A dolgozó fizetése'+60000 > (SELECT MAX('A dolgozó fizetése') FROM Dolgozó)
```

A keletkezett eredménytábla az alábbi:

A dolgozó neve	A dolgozó fizetése
Nagy József	150000
Vári Ödön	210000

Példa

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

Az alábbi parancsot használhatjuk:

```
SELECT 'A dolgozó neve', 'A dolgozó törzsszáma' FROM Dolgozó
WHERE 'A dolgozó törzsszáma' NOT IN (SELECT 'A dolgozó törzsszáma'
FROM Kifizetés)
```

Egy másik lehetséges megoldás:

```
SELECT 'A dolgozó neve', 'A dolgozó törzsszáma' FROM Dolgozó
WHERE NOT EXISTS (SELECT 'A dolgozó törzsszáma' FROM Kifizetés
WHERE Kifizetés.'A dolgozó törzsszáma' = Dolgozó.'A dolgozó
törzsszáma')
```

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

A keletkezett eredménytábla az alábbi:

A dolgozó neve	A dolgozó törzsszáma
Vári Ödön	T768545
Kovács János	T429877

Példa

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

Az alábbi parancsot használhatjuk:

SELECT * FROM Dolgozó WHERE 'A dolgozó fizetése' > ALL (SELECT 'A dolgozó fizetése' FROM Dolgozó WHERE 'A dolgozó születési helye' LIKE "Eger" OR 'A dolgozó születési helye' LIKE "Szeged")

A dolgozó törzsszáma	A dolgozó neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T456734	Nagy József	Budapest	1972. 01. 30.	150000
T768545	Vári Ödön	Budapest	1958. 07. 12.	210000

Az adatbázis módosítása SQL segítségével

Relációsémák definiálása

Definiálhatjuk a tábla nevét, megadhatjuk az attribútumait, azok típusát és méretét. A legfontosabb adattípusok, amelyeket az SQL szabvány definiál:

Egész számok. Ezek megadásánál a **SHORTINT**, **INT** vagy **INTEGER** kulcsszavakat használhatjuk.

Valós számok, lebegőpontos tárolással. Ezeknél is különböző méretben tárolt számokat adhatunk meg.

A **FLOAT** és a **REAL** a hagyományos programozás nyelvekből is ismert normál lebegőpontos számot jelenti. **A DOUBLE PRECISION** duplapontos számot jelent, míg speciálisan az SQL-ben használhatjuk a **DECIMAL(<számjegy>,<tizedesjegy>)** formát is, ahol explicit módon megadhatjuk, hogy a szám hány számjegyből állhat, illetve hány tizedesjegyet tartalmazhat.

Fix vagy változó hosszúságú karaktersorozatok.

Megadásuk a **CHAR(<hossz>)**, illetve a **VARCHAR(<hossz>)** paranccsal történik. A **CHAR** segítségével olyan attribútumot definiálhatunk, amely pontosan a megadott hosszúságú karaktersorozatként fogja tárolni az

adatokat, míg a **VARCHAR**-ral megadott attribútumoknál csak az aktuális számú karakter kerül tárolásra.

Dátum és idő. Ezeket a **DATE** és **TIME** kulcsszavakkal lehet megadni.

A táblák létrehozására irányuló parancs szintaxisát.

CREATE TABLE <táblanév> {<attribútumdefiníció> [,<attribútumdefiníció>]… }

Az <attribútumdefiníció> paraméterben adjuk meg az egyes attribútumok nevét és típusát, a következő módon:

<név> <típus> [DEFAULT <érték>]

A <név> jelenti az attribútum nevét, míg a <típus> adja meg a típust és a méretet, a fentiekben ismertetett módon. A **DEFAULT** paranccsal alapértelmezett értékeket adhatunk meg az egyes attribútumoknak, ennek használata nem kötelező.

Lehetőség van arra is, hogy már meglévő táblához új oszlopot adjunk. Ez a következőképpen történhet:

ALTER TABLE <táblanév> ADD <attribútumdefiníció>

Hasonlóan történhet egy oszlop eltávolítása a táblából. Ebben az esetben az attribútum típusát nem szükséges megadni, elegendő a neve.

ALTER TABLE <táblanév> DROP <attribútumnév>

Lehetőségünk van tábla törlésére is a következő módon:

DROP <táblanév>

Tudjuk, hogy egy tábla definiálásakor a tábla nevén és az attribútumokon túlmenően még egyéb információkat is meg kell adnunk.

Ilyenek a **kulcsok**, az attribútum értékekre vonatkozó **korlátozások**.

Ha elsődleges kulcsot szeretnénk az SQL nyelv segítségével definiálni, a tábla létrehozását kibővíthetjük speciális parancsokkal.

Ez a következőképpen néz ki:

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

A <kulcsattribútum> paraméterben kell megadni annak az attribútumnak a nevét, amely a kulcsot alkotja, vagy annak egy részét képezi.

Amennyiben csak egy attribútum tartozik a kulcshoz, akkor használhatjuk mind a **PRIMARY KEY**, mind a **UNIQUE** parancsokat. Több attribútumból álló kulcsot csak a **PRIMARY KEY** kulcsszóval definiálhatunk. A **UNIQUE** kulcsszó segítségével minden egyes attribútumnál megadhatjuk, hogy az adott attribútum csak egyedi értékeket vehet fel.

Hasonlóan az elsődleges kulcs megadásához, az SQL lehetőséget biztosít **idegen kulcsok** definiálására is.

Ennek módja az alábbi:

CREATE TABLE <táblanév> { <attribútumdefiníció> [REFERENCES <táblanév> (<attribútumnév>)] [,<attribútumdefiníció> [REFERENCES <táblanév> (<attribútumnév>)]]… [,FOREIGN KEY (<kulcsattribútum> [,<kulcsattribútum>]…)<táblanév>(<kulcsattribútum>[,<kulcsattribútum>]…)]}

Láthatjuk, hogy az idegen kulcs megadása teljesen hasonló az elsődleges kulcshoz.

A különbség mindössze annyi, hogy idegen kulcsnál mindig meg kell adni, hogy az attribútum melyik másik tábla melyik kulcsmezőjéhez kapcsolódik.

Amennyiben az idegen kulcs egy attribútumból áll, használhatjuk a **REFERENCES** kulcsszót,

ha azonban az idegen kulcs összetett, akkor a **FOREIGN KEY** kulcsszóval kell definiálnunk.

Említettük, hogy az ABKR-nek gondoskodnia kell az úgynevezett hivatkozási épség fenntartásáról. Ez azt jelenti, hogy ha egy idegen kulcsban hivatkozunk egy másik tábla egy kulcsértékére, akkor a megadott értékű **előfordulásnak létezni kell**.

Amennyiben olyan **módosító, vagy törlő műveletet** hajtunk végre, ami ezt a szabályt megsérti, akkor az ABKR-nek ezt valahogyan kezelni kell.

Egyik lehetőség, hogy a műveletet **nem engedi** végrehajtani, a másik, hogy **megengedi,** de a hivatkozási épség fenntartása érdekében **automatikusan korrigálja** az adatbázist. A korrigálás kétféleképpen történhet.

Az egyik az, hogy ha egy hivatkozott sort törlünk vagy módosítunk, akkor a rá hivatkozó előfordulások is törlődnek, vagy módosulnak a másik táblában.

A másik lehetőség az, hogy a helytelen hivatkozásokat egy speciális értékkel korrigálja az ABKR, amelyet nevezhetünk NULL értéknek.

Amennyiben azt szeretnénk, hogy valamely korrigáló eljárás lépjen életbe a hivatkozási épség megsérülésekor, akkor ezt a tábla definiálásakor az SQL parancsban külön megadhatjuk.

Mivel a hivatkozási épség mindig idegen kulcsokra vonatkozik, ezért ezt csak ezeknél lehet használni. A szintaxis definíciójánál a **REFERENCES** kulcsszóval megadott részt egészítjük ki.

REFERENCES <táblanév> (<attribútumnév>) [[ON DELETE SET NULL|CASCADE] [ON UPDATE SET NULL|CASCADE]]

Az **ON DELETE** részben azt adhatjuk meg, hogy a törlés során bekövetkezett hivatkozás épség sérülését hogyan kezelje a rendszer,

míg

az **ON UPDATE** részben a módosításkor bekövetkezőt. Mindkét esetben az ismertetett két lehetőség közül választhatunk, vagyis a NULL érték beállítása, **(SET NULL)**, illetve korrigálás a hivatkozó táblában is **(CASCADE).**

A tábla definiálásakor megadható megszorítások következő nagy csoportját az **attribútumok értékére vonatkozó korlátozások** alkotják. Ennek legegyszerűbb fajtája az, amikor a megszorítás az egyes attribútumok értékeire vonatkozik.

Ezt szintén a tábla definiálásakor adhatjuk meg, az attribútum leírásakor. Ennek formája a következő:

<attribútumdefiníció> NOT NULL|CHECK (<feltétel>)

A **NOT NULL** opció azt jelenti, hogy az adott attribútum nem vehet fel NULL értéket.

A CHECK kulcsszó után **tetszőleges feltételt** adhatunk. Az erre vonatkozó szabályok megegyeznek a WHERE SQL parancs után használt feltétel megadásánál alkalmazottakkal.

A feltétel ellenőrzése sor beszúrásakor, vagy az attribútum módosításakor történik.

Formája az alábbi:

CREATE TABLE <táblanév> { <attribútumdefiníció> [,<attribútumdefiníció>]… [CHECK <feltétel>]}

Példa

Adjuk meg azt az SQL parancsot , amely létrehozza a Dolgozó táblát!

CREATE TABLE Dolgozó {'A dolgozó törzsszáma' CHAR(7), 'A dolgozó neve' VARCHAR(50), 'A dolgozó születési helye' VARCHAR(30) 'A dolgozó születési ideje' DATE, 'A dolgozó fizetése' DECIMAL(7,0) }

Példa

Adjuk meg azt az SQL parancsot, amely a Dolgozó táblát létrehozza, és elsődleges kulcsnak a A dolgozó törzsszáma mezőt definiálja!

CREATE TABLE Dolgozó {'A dolgozó törzsszáma' CHAR(7) PRIMARY KEY, 'A dolgozó neve' VARCHAR(50), 'A dolgozó születési helye' VARCHAR(30) 'A dolgozó születési ideje' DATE, 'A dolgozó fizetése' DECIMAL(7,0) }

Egy másik lehetséges megoldás:

CREATE TABLE Dolgozó {'A dolgozó törzsszáma' CHAR(7), 'A dolgozó neve' VARCHAR(50), 'A dolgozó születési helye' VARCHAR(30) 'A dolgozó születési ideje' DATE, 'A dolgozó fizetése' DECIMAL(7,0) PRIMARY KEY ('A dolgozó törzsszáma')}

Példa

Tegyük fel, hogy a Kifizetés táblában található **'A dolgozó törzsszáma'** nevű mező idegen kulcs, amely a **Dolgozó** táblával való kapcsolatot valósítja meg. Adjuk meg azt az SQL parancsot, amely a fenti **Kifizetés** táblát létrehozza, úgy hogy a hivatkozási épség sérülésekor az ABKR azt automatikusan frissítéssel korrigálja!

CREATE TABLE Kifizetés {'A kifizetés dátuma' DATE, 'A kifizetett bér' DECIMAL(7,0), 'A levont adóelőleg' DECIMAL(7,0) 'A dolgozó törzsszáma' CHAR(7) REFERENCES Dolgozó ('A dolgozó törzsszáma') ON UPDATE CASCADE ON DELETE CASCADE}

Egy másik lehetséges megoldás:

CREATE TABLE Kifizetés {'A kifizetés dátuma' DATE, 'A kifizetett bér' DECIMAL(7,0), 'A levont adóelőleg' DECIMAL(7,0) 'A dolgozó törzsszáma' CHAR(7), FOREIGN KEY ('A dolgozó törzsszáma') Dolgozó ('A dolgozó törzsszáma') ON UPDATE CASCADE ON DELETE CASCADE}

Példa

Adjuk meg azt az SQL parancsot, amely a Dolgozó táblát úgy definiálja, hogy a 'A dolgozó fizetése' mező esetén mindig ellenőrzésre kerüljön, hogy az éppen megadott érték eléri-e egy minimális bér összegét, mondjuk 50000 Ft-ot!

CREATE TABLE Dolgozó {'A dolgozó törzsszáma' CHAR(7), 'A dolgozó neve' VARCHAR(50), 'A dolgozó születési helye' VARCHAR(30) 'A dolgozó születési ideje' DATE, 'A dolgozó fizetése' DECIMAL(7,0) CHECK 'A dolgozó fizetése' >50000}

Példa

Adjuk meg azt az SQL parancsot , amely hozzáadja a Dolgozó táblához a dolgozó lakcímét!

ALTER TABLE Dolgozó ADD 'A dolgozó lakcíme' VARCHAR(50)

Példa

Adjuk meg azt az SQL parancsot, amely a Dolgozó táblából törli A dolgozó születési helye attribútumot!

ALTER TABLE Dolgozó DROP 'A dolgozó születési helye'

Változtatások az adatbázisban (DML)

Új sorok beszúrása a következő paranccsal történhet:

INSERT INTO <relációséma> VALUES (<értéklista>)

A <relációséma> paraméterben meg kell adnunk a reláció nevét. majd zárójelben azokat az attribútum neveket, amelyekhez az **<értéklista>** paraméterben megadott egymástól vesszővel elválasztott értékeket rendeljük. Amennyiben csak a reláció nevét adjuk meg, azt alapértelmezésben úgy tekinti a rendszer, mintha az összes attribútumot felsoroltuk volna.

A két lista elemeinek számban, sorrendben és típusban meg kell egyezniük. A végrehajtás során a megfelelő értékek hozzárendelődnek a megfelelő attribútumhoz, és az így keletkezett sor bekerül az adattáblába. Amennyiben egy létező attribútum név nem szerepel a listában, akkor ahhoz az alapértelmezés szerinti értéke rendelődik, amennyiben ilyen van, vagy pedig nem kap értéket, azaz például a **NULL** érték kerül bele.

A beszúrás művelet ellentettje a **törlés**. Ennél a specialitást az jelenti, hogy meg kell adnunk azt a feltételt, amelynek eleget tevő sorokat törölni szeretnénk. Ennek formája az alábbi:

DELETE FROM <reláció> WHERE <feltétel>

A <feltétel> paraméterben a már jól ismert módon adhatjuk meg azt a feltételt, amely alapján a törlés történik.

A módosítás szintaxisa a következő:

UPDATE <reláció> SET <értékadások> WHERE <feltétel>

Az értékadások vesszővel vannak egymástól elválasztva. Mindegyik értékadás egy attribútum névből, egyenlőségjelből és értékből áll. Az értékek módosítása azon sorokra történik meg, amelyek a megadott feltételnek eleget tesznek.

Példa

Adjuk meg azt az SQL parancsot, amely egy teljes sort szúr be a táblába!

INSERT INTO Dolgozó VALUES ("T768545","Vári Ödön", "Budapest", 1958.07.12. 210000)

Az előző példában szereplő Dolgozó táblára vonatkozóan adjuk meg azt az SQL parancsot, amely egy olyan dolgozót szúr be a táblába, akinek egyelőre csak a neve és a törzsszáma áll rendelkezésre!

INSERT INTO Dolgozó('A dolgozó törzsszáma', 'A dolgozó neve' VALUES ("T429877","Kovács János")

Példa

Adjuk meg azt az SQL parancsot, amely a Dolgozó táblából törli az összes Budapesten született dolgozót!

```
DELETE FROM Dolgozó WHERE 'A dolgozó születési helye' = "Budapest"
```

Példa

Adjuk meg azt az SQL parancsot, amely beírja a hiányzó adatokat a fenti Dolgozó táblába!

```
UPDATE Dolgozó SET 'A dolgozó születési helye' = "Szeged", 'A dolgozó születési ideje' = 1967.05.12., 'A dolgozó fizetése' = 120000 WHERE 'A dolgozó törzsszáma'="T429877"
```