

# Adatbázis fejlesztés és üzemeltetés II.

Szabó Bálint

## MÉDLAINFORMATIKAI KIADVÁNYOK

# Adatbázis fejlesztés és üzemeltetés

## II.

Szabó Bálint



Eger, 2013



Korszerű információtechnológiai szakok magyarországi adaptációja

**TÁMOP-4.1.2-A/1-11/1-2011-0021**

Nemzeti Fejlesztési Ügynökség  
[www.ujszechenyiterv.gov.hu](http://www.ujszechenyiterv.gov.hu)  
06 40 638 638



A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

*Lektorálta:*

Nyugat-magyarországi Egyetem Regionális Pedagógiai Szolgáltató és  
Kutató Központ

Felelős kiadó: dr. Kis-Tóth Lajos

Készült: az Eszterházy Károly Főiskola nyomdájában, Egerben

Vezető: Kérészy László

Műszaki szerkesztő: Nagy Sándorné

# Tartalom

<b>1.</b>	<b><i>Bevezetés .....</i></b>	<b><i>11</i></b>
1.1	Célkitűzések, kompetenciák a tantárgy teljesítésének feltételei .....	11
1.1.1	Célkitűzés.....	11
1.1.2	Kompetenciák.....	12
1.1.3	A tantárgy teljesítésének feltételei .....	12
1.2	A kurzus tartalma .....	13
1.3	Tanulási tanácsok, tudnivalók .....	14
1.4	Források.....	16
<b>2.</b>	<b><i>Lecke: MySQL-adatbázisok kezelése.....</i></b>	<b><i>17</i></b>
2.1	Célkitűzések és kompetenciák .....	17
2.2	Adat, információ kommunikáció .....	17
2.3	Kommunikáció .....	18
2.4	Adatbázis-rendszerek .....	19
2.5	Adatmodell .....	21
2.6	Gyakorló adatbázis.....	22
2.7	MySQL adatbázis-kezelő rendszer .....	23
2.7.1	Az SQL .....	24
2.7.2	Az SQL nyelvtana .....	25
2.8	Kapcsolódás MySQL-szerverhez .....	27
2.8.1	A MySql parancsori kliense.....	28
2.9	Munkamenet beállítása .....	32
2.10	Összefoglalás, kérdések .....	34
2.10.1	Összefoglalás .....	34
2.10.2	Önellenőrző kérdések.....	35
<b>3.</b>	<b><i>Lecke: Választó lekérdezések készítése.....</i></b>	<b><i>37</i></b>
3.1	Célkitűzések és kompetenciák .....	37
3.2	A SELECT parancs általános formája .....	38
3.3	Példák a SELECT parancs alkalmazásaira.....	39
3.3.1	SELECT használata értékek kiírására.....	40

3.3.2	Rekordforrás és mezők megadása .....	40
3.3.3	Álnevek az eredményhalmazban .....	42
3.3.4	Számított mezőértékek .....	43
3.3.5	Rekordok kiválasztása .....	45
3.3.6	Rekordok sorba rendezése .....	48
3.3.7	Különböző rekordok kiválogatása .....	49
<b>3.4</b>	<b>Összefoglalás, kérdések .....</b>	<b>49</b>
3.4.1	Összefoglalás .....	49
3.4.2	Önellenőrző kérdések.....	50
<b>4.</b>	<b><i>Lecke: Kapcsolódó táblák rekordjainak kezelése.....</i></b>	<b>53</b>
<b>4.1</b>	<b>Célkitűzések és kompetenciák .....</b>	<b>53</b>
<b>4.2</b>	<b>Táblák közötti kapcsolatok .....</b>	<b>54</b>
<b>4.3</b>	<b>Több táblából álló rekordforrások .....</b>	<b>55</b>
4.3.1	Szoros illesztés.....	57
<b>4.4</b>	<b>Laza illesztésű kapcsolatok .....</b>	<b>60</b>
<b>4.5</b>	<b>Összefoglalás, kérdések .....</b>	<b>62</b>
4.5.1	Összefoglalás .....	62
4.5.2	Önellenőrző kérdések.....	63
<b>5.</b>	<b><i>Lecke: Függvények használata az adatkezelésben .....</i></b>	<b>67</b>
<b>5.1</b>	<b>Célkitűzések és kompetenciák .....</b>	<b>67</b>
<b>5.2</b>	<b>A függvényekről általában.....</b>	<b>67</b>
<b>5.3</b>	<b>Szövegkezelő függvények.....</b>	<b>68</b>
5.3.1	Szöveg csonkolása .....	68
5.3.2	Szöveg módosítása .....	70
5.3.3	Szöveg keresése.....	71
5.3.4	Szöveg formázása .....	72
5.3.5	Szöveg hosszának meghatározása.....	72
<b>5.4</b>	<b>Dátum- és időkezelő függvények .....</b>	<b>73</b>
5.4.1	Munka a rendszeridővel .....	74
5.4.2	Dátum és idő részeinek feldolgozása .....	74
5.4.3	UNIX_TIMESTAMP .....	75
5.4.4	Műveletek dátumokkal .....	76
5.4.5	Dátumok formátumának szabályozása .....	77
<b>5.5</b>	<b>Matematikai függvények.....</b>	<b>79</b>
5.5.1	Egyszerűbb matematikai számítások .....	79

5.5.2	Számok előjelének kezelése és kerekítés .....	80
5.5.3	Véletlenszámok generálása .....	81
5.6	Függvények használata a lekérdezésekben .....	82
5.7	Összefoglalás, kérdések .....	84
5.7.1	Összefoglalás .....	84
5.7.2	84	
5.7.3	Önellenőrző kérdések .....	85
6.	<b>Lecke: Statisztikai számítások SQL-ben .....</b>	<b>87</b>
6.1	Célkitűzések és kompetenciák .....	87
6.2	Összesítő lekérdezések .....	87
6.2.1	Összesítő függvények .....	89
6.3	A MySQL összesítő függvényei .....	90
6.4	Részhalmazok kialakítása az összesítő lekérdezésekben .....	92
6.5	Rekordok kiválogatása .....	95
6.6	WITH ROLLUP módosító .....	96
6.7	Összesítő lekérdezések gyakorlati alkalmazása .....	97
6.8	Összefoglalás, kérdések .....	101
6.8.1	Összefoglalás .....	101
6.8.2	Önellenőrző kérdések .....	102
7.	<b>Lecke: Allekérdezések használata .....</b>	<b>103</b>
7.1	Célkitűzések és kompetenciák .....	103
7.2	Fő- és allekérdezés .....	104
7.3	Allekérdezés beillesztése .....	105
7.4	Allekérdezések helye a főlekérdezésben .....	106
7.4.1	Allekérdezés WHERE záradékban .....	107
7.4.2	Allekérdezés a rekordforrásban .....	111
7.4.3	Allekérdezések mező meghatározásban .....	116
7.5	Összefoglalás, kérdések .....	117
7.5.1	Összefoglalás .....	117
7.5.2	Önellenőrző kérdések .....	118
8.	<b>Lecke: Rekordok beszúrása az adatbázis tábláiba .....</b>	<b>121</b>
8.1	Célkitűzések és kompetenciák .....	121

8.2	<b>Az INSERT parancs általános formája .....</b>	<b>122</b>
8.3	<b>Rekordok beszúrásának szabályai.....</b>	<b>123</b>
8.3.1	Táblaterv megtekintése.....	124
8.4	<b>Az INSERT opcionális elemei.....</b>	<b>125</b>
8.5	<b>Rekordok beszúrása mezőértékek felsorolásával .....</b>	<b>125</b>
8.5.1	Rekord beszúrása az összes mezőérték felsorolásával. ....	126
8.5.2	Rekord beszúrása az mezőnevek és mezőérték felsorolásával.....	129
8.5.3	Több rekord beszúrása:.....	129
8.6	<b>Rekordok megadása allekérdezésekkel.....</b>	<b>130</b>
8.7	<b>Rekordok eredményhalmazának fájlba mentése.....</b>	<b>132</b>
8.8	<b>Szövegfájl tartalmának betöltése az adatbázisba .....</b>	<b>133</b>
8.9	<b>Összefoglalás, kérdések .....</b>	<b>135</b>
8.9.1	Összefoglalás .....	135
8.9.2	Önellenőrző kérdések.....	136
9.	<b><i>Lecke: Rekordok módosítása, és eltávolítása .....</i></b>	<b><i>139</i></b>
9.1	<b>Célkitűzések és kompetenciák.....</b>	<b>139</b>
9.2	<b>Alapfokú adatbiztonság .....</b>	<b>140</b>
9.3	<b>Rekordok frissítése .....</b>	<b>141</b>
9.3.1	Többfelhasználós környezet.....	142
9.3.2	Rekordforrás magadása .....	143
9.3.3	Mezőértékek frissítése .....	143
9.3.4	Rekordok kiválasztása .....	144
9.3.5	Sorrend és rekordszám.....	146
9.4	<b>Rekordok törlése .....</b>	<b>149</b>
9.5	<b>Tábla kiürítése .....</b>	<b>151</b>
9.6	<b>Összefoglalás, kérdések .....</b>	<b>152</b>
9.6.1	Összefoglalás .....	152
9.6.2	Önellenőrző kérdések.....	153
10.	<b><i>Lecke: A MySQL továbbfejlesztett lehetőségei .....</i></b>	<b><i>155</i></b>
10.1	<b>Célkitűzések és kompetenciák.....</b>	<b>155</b>
10.2	<b>Tranzakciók kezelése.....</b>	<b>155</b>
10.2.1	A tranzakciók a MySQL-ben: .....	157



10.3	Nézetek.....	160
10.4	Tárolt eljárások .....	162
10.4.1	Tárolt eljárások létrehozása .....	163
10.4.2	Tárolt eljárás paraméterei .....	163
10.4.3	Vezérlési szerkezetek.....	164
10.4.4	Ciklusok.....	165
10.4.5	Eljárások hívása .....	165
10.5	Tárolt függvények.....	166
10.5.1	Függvény deklarációja .....	167
10.6	Triggerek.....	168
10.7	Összefoglalás, kérdések .....	171
10.7.1	Összefoglalás .....	171
10.7.2	Önellenőrző kérdések.....	172
11.	<b>Lecke: MySQL DBMS és adatbázis alkalmazások kommunikációja .....</b>	<b>173</b>
11.1	Célkitűzések és kompetenciák .....	173
11.2	A MySQL protokoll implementációi.....	173
11.3	MySQL ODBC connector használata .....	175
11.4	MS Access adatbázis kapcsolása ODBC-adatforráshoz.....	177
11.5	PHP MySQL API .....	181
11.5.1	MySQL API használata .....	182
11.6	Összefoglalás, kérdések .....	187
11.6.1	Összefoglalás .....	187
11.6.2	Önellenőrző kérdések.....	187
12.	<b>Összefoglalás.....</b>	<b>189</b>
12.1	Tartalmi összefoglalás .....	189
12.1.1	Bevezetés.....	189
12.1.2	MySQL-adatbázisok kezelése.....	189
12.1.3	Választó lekérdezések készítése .....	189
12.1.4	Kapcsolódó táblák rekordjainak kezelése.....	189
12.1.5	Függvények használata az adatkezelésben .....	189
12.1.6	Statisztikai számítások SQL-ben .....	190
12.1.7	Allekérdezések használata.....	190
12.1.8	Rekordok beszúrása az adatbázis tábláiba .....	190
12.1.9	Rekordok módosítása, és eltávolítása .....	190

12.1.10 A MySQL továbbfejlesztett lehetőségei .....	190
12.1.11 Adatok megjelenítése különböző platformokon.....	190
12.1.12 Összefoglalás .....	191
<b>12.2 Fogalmak.....</b>	<b>191</b>
12.2.1 Bevezetés .....	191
12.2.2 MySQL-adatbázisok kezelése .....	191
12.2.3 Választó lekérdezések készítése .....	191
12.2.4 Kapcsolódó táblák rekordjainak kezelése .....	191
12.2.5 Függvények használata az adatkezelésben .....	191
12.2.6 Statisztikai számítások SQL-ben .....	191
12.2.7 Allekérdezések használata .....	191
12.2.8 Rekordok beszúrása az adatbázis tábláiba .....	191
12.2.9 Rekordok módosítása, és eltávolítása .....	192
12.2.10 A MySql továbbfejlesztett lehetőségei.....	192
12.2.11 : MySQL DBMS és adatbázis alkalmazások kommunikációja .....	192
<b>12.3 Zárás</b>	<b>192</b>
<b>13. Kiegészítések.....</b>	<b>193</b>
<b>13.1 Irodalomjegyzék .....</b>	<b>193</b>
13.1.1 Hivatkozások.....	193

# 1. BEVEZETÉS

## 1.1 CÉLKITÚZÉSEK, KOMPETENCIÁK A TANTÁRGY TELJESÍTÉSÉNEK FELTÉTELEI

A XXI. század embere magánéletének és munkájának egyaránt szerves részévé vált az informatika. Az alig egy-két évtizede még meglehetősen bonyolultnak számító, és sokszor csak az „informatikai guruk” munkaeszközüül szolgáló hardver- és szoftverberendezések hatalmas átalakuláson mentek keresztül. Míg korábban a viszonylag egyszerű feladatok elvégzéséhez is kiterjedt tudásra volt szükség, napjaink informatikai eszközei jóval egyszerűbb, tetszetősebb, barátságosabb felülettel rendelkeznek. A velük végzett munka során már nem feltétlenül kell koncentrálnunk a hardverre, a fájlrendszer működésére, az éppen használt szolgáltatást biztosító alkalmazás programkódjára, vagy a feldolgozott adatok fizikai tárolásának módjára. Az átlagos, „naiv” felhasználónak nem kell foglalkoznia a háttérben meghúzódó folyamatokkal, az informatikai szakembernek azonban tudnia kell, hogy a látszólagos egyszerűség varázsa mögött változatlanul ott bújik az egyre nagyobb teljesítményt biztosító hardver, az informatikai eszköz alapszolgáltatásait biztosító operációs rendszer, az adott feladat lebonyolítását lehetővé tevő felhasználói program, és ott húzódnak a munkánk során használt adatok tárolását megvalósító, különböző bonyolultságú adatbázisok is.

Ilyen adatbázisokat használunk, amikor telefonunkban előkeressük egy partner számát, amikor weblapok után kutatunk valamelyik keresőrendszerrel, vásárlunk az interneten, vagy cégünk termékei között tallózunk. A meglehetősen bonyolult hardverhez hasonlóan az adatbázisok, illetve az adatbázis-kezelő rendszerek működését is „eltakarják” a felhasználói felületek. Míg azonban a felhasználónak általában nem, addig informatikusnak pontosan kell ismernie az adatbázisok szerkezetét, tervezésük és kialakításuk lépéseit, a használatukban rejlő lehetőségeket, és kezelésük módszereit.

Tankönyvsorozatunk kapcsolódó kötetében az adatbázisok tervezésével és megvalósításával ismerkedhet meg az olvasó. Jelen tananyag az elkészült relációs adatbázisok kezelésével, az adatok tárolásával, manipulálásával, az adatbázis alapján történő informálódás lehetőségeivel foglalkozik.

### 1.1.1 Célkitűzés

Tananyagunk megírása közben azt a célt tartottuk szem előtt, hogy a lekéket figyelmesen átolvasó, tanuló a „naiv”, eseti felhasználó tudását messze

meghaladó, a szakértő felhasználó ismeretein is túlmutató tudásra tehesen szert. Reményeink szerint a tananyagunkban foglalt ismereteket elsajátító olvasó tudása, az adatbázis-menedzser felhasználói szerepkörnek megfelelő szintre emelkedik, és rendelkezik majd mindazokkal az ismeretekkel, amelyek birtokában egyszerű karakteres kliensek, vagy pedig külső alkalmazások segítségével, hatékonyan tudja kezelni a mások, vagy akár saját maga által készített MySQL-adatbázisokat.

### **1.1.2 Kompetenciák**

A tananyag leckéinek elolvasása után Ön képes lesz:

- adatbázis alkalmazással, vagy egyszerű mysql-klienssel a MySQL adatbázis-kezelő rendszerekhez kapcsolódni,
- a karakteres klienst beállítani, a kapcsolat munkamenetét konfigurálni
- a kezelni kívánt adatbázist kiválasztani,
- egy vagy több egymáshoz kapcsolódó tábla rekordjait lekérdezni,
- az adatbázisban tárolt adatokon alapuló statisztikákat készíteni,
- allekérdezések használatával összetett lekérdezéseket készíteni,
- a táblában tárolt adatokat frissíteni és szükség esetén törölni,
- tárolt eljárásokat, függvényeket készíteni,
- adatbázisban bekövetkező eseményekhez triggereket rendelni,
- az adatbázis-kezelő rendszer felügyelete alá tartozó adatokat kinyerni, és külső alkalmazásokkal fölhasználni.

### **1.1.3 A tantárgy teljesítésének feltételei**

Tananyagunk megírásakor feltételeztük, hogy az olvasó ismeri a relációs adatbázis-kezelés alapfogalmait, a relációs adatbázisok tervezésének és kialakításának lépéseit. Képes az adatbázisok koncepcionális, logikai, fizikai megtervezésére, valamint létrehozására. Jelen tananyag egységes sorozatba illeszkedik. E sorozat tematikáját tekintve előző eleme, az Adatbázis-fejlesztés és -üzemeltetés I. éppen ezzel a témakörrel foglalkozik, így a tárgy sikeres teljesítésének feltételül e tananyag ismeretét jelölhetjük meg.

Mindamellett szándékunk olyan ismeretanyag összeállítása, amely az adatbázisok tervezését, és fizikai kialakítását csak alapszinten ismerő tanulók számára is megtanulható és elsajátítható formában foglalja össze az adatbázisok használatával kapcsolatos ismereteket.

## 1.2 A KURZUS TARTALMA

A tananyag összesen 12 leckére tagolódik. A most olvasott 1. lecke bevezető információkkal, és tanulási tanácsokkal szolgál, az utolsó, 12. lecke pedig összegzi a tananyagban megszerezhető ismereteket. A 2–11. leckék tartalmazzák a korábban felsorolt kompetenciák megszerzéséhez szükséges ismeretanyagot.

- A 2. lecke áttekinti a tananyag megértéshez szükséges fogalmakat, szól a MySQL-adatbázisokhoz való kapcsolódás lehetőségeiről, és a a kliens, és munkamenetek beállításairól.
- A 3. leckében az SQL DQL elemének alapvető parancsával, a választó lekérdezések készítésére alkalmas SELECT-tel ismerkedhet meg.
- A 3. lecke módszerével egyszerre egy táblából tud majd rekordokat kiválogatni.
- A 4. lecke mutatja be, hogyan használhat több, egymáshoz kapcsolódó táblát, a választó lekérdezések rekordforrásaként. Itt ismerkedhet meg a laza, és szoros illesztések fogalmával, és gyakorlati megvalósításával.
- Lekérdezések készítésekor gyakori, hogy az eredményhalmazban nem közvetlenül a tárolt adatokat, hanem azok valamilyen módosított formáját, vagy a velük végzett számítás eredményét akarjuk megjeleníteni. Ilyenkor tehetnek jó szolgálatot az adatbázis-kezelő rendszer beépített számításai, az úgynevezett függvények. Az 5. lecke a MySQL dátum-, szöveg- és matematikai függvényei közül mutatja be a legfontosabbakat.
- A 6. lecke a választó lekérdezések egy speciális fajtája, a statisztikai műveletek elvégzését biztosító, összesítő lekérdezések készítésének technikájával ismerteti meg az olvasót.
- Bonyolultabb adatbázis-kezelő feladatok során előfordulhat, hogy egy lekérdezés valamely elemét, egy másik, úgynevezett allekérdezés is biztosíthatja. A 7. leckében az allekérdezések mibenlétéről, előnyeiről és kialakításáról olvashat.
- A 8. lecke az SQL DML első elemét, az új rekordokat beszúrását biztosító INSERT parancs használatát mutatja be. A lecke elsajátításával megtanulhatja, hogyan lehet egyetlen, vagy több rekordot, esetleg egy választó lekérdezés eredményhalmazát beszúrni egy meglévő táblába.
- A 9. lecke szintén DML-elemeket, a rekordok módosítására használt UPDATE, és a törlésre alkalmas, DELETE parancsokat ismerteti.

- A 10. leckében olyan speciális lehetőségekről tanulhat, mint a párhuzamos hozzáférés esetén szükséges tranzakciókezelés, a nézetek, valamint a triggerek, és tárolt eljárások használata.
- A 11. lecke a MySQL-adatbázisok külső alkalmazásokból történő használatára, például a Microsoft Access felületéről történő elérés biztosítására mutat példát. Ebben a leckében ismerkedhet meg a webfelületen működő adatbázis-alkalmazások készítésének alapjaival is.

## 1.3 TANULÁSI TANÁCSOK, TUDNIVALÓK

Tananyagunk 2–11. leckéje felöleli az adatbázis menedzsment témakör szükséges ismereteit. A leckék azonos szerkezetűek, fölépítésüket úgy igyekeztünk kialakítani, hogy a lehető legjobban segítsék az olvasót a megértésben, és a tananyag eredményes elsajátításában.

Minden lecke a **Célkitűzés, kompetenciák** szakasszal kezdődik. Ebben a bevezetésként is felfogható leckerészben találja meg az anyag áttanulmányozásával megszerezhető kompetenciákat, illetve itt olvashat a kitűzött célokról is. Célok alatt ne egyszerű felsorolást képzeljen el. Általában olyan problémákat, kérdéseket vetünk fel, amelyek az előző fejezetek alapján már Önben is megfogalmazódhattak. A lecke célja, hogy az új ismeretekkel megkeressük, és meg is adjuk a válaszokat a felsorolt problémákra. A bevezető kérdések ennek megfelelően, a lecke logikai gondolatmenetét is meghatározzák. Arra kérjük, gondolkodjon együtt a tananyag írójával. A szöveg olvasásakor keresse a válaszokat, és ne lépjen tovább a leckéből, amíg azokat meg nem találta.

A célok után a lecke ismeretanyaga következik. A szövegben eltérő formátummal jeleztük a valamilyen szempontból kiemelkedő bekezdéseket, szövegrészeket. Az alábbi formátumokkal találkozhat:

**Alkalmazások menüelemei, menüparancsok**

**Definíciók, fogalmak**

**Fájlrendszerben használt elérési utak**

Feladatok

**Fontos szöveg**

**Grafikus felületen található vezérlő elemek, objektumok**

 Gyakorlatok

**Kódok, SQL-mondatok****Megjegyzések****Nyomógombok, forró billentyűk****Összefoglaló kérdések**

- **Válaszok**

A leckében található **fogalmakat, definíciókat** igyekezzen a legpontosabban megtanulni. Természetesen nem a szó szerinti ismétlés, hanem a lényeg szabatos megfogalmazása a fontos.

Fordítson különös figyelmet a **fontos** szövegrészekre!

A **gyakorlatokat, feladatokat** minden esetben végezze el. Ezek ugyanis hozzásegítik ahhoz, hogy a szerzett ismereteket a gyakorlatban is képes legyen kamatoztatni.

A **kódokat**, SQL-mondatokat elsősorban a személtetés érdekében illesztettük be, azonban úgy igyekeztünk elhelyezni őket a tananyagban, hogy vágólapon keresztül, közvetlenül is bemásolhatók legyenek a felhasználás helyére.

**Ha a kódok felhasználásának ezt a módját választja, legyen óvatos!**

A vágólapról történő beillesztés során gyakran jelentkeznek kisebb-nagyobb hibák, melyek a karakterek konverziójából adódnak. Tipikusan ilyen az idézőjelek fölcserélődése, vagy a sorvégjelek okozta hibák. Mielőtt futtatja a vágólappal másolt kódokat, minden esetben végezzen szintaktikai ellenőrzést!

Minden egyes lecke végén megtalálja **Összefoglalás** szakaszt, ami logikusan követhető sorrendbe szedve, tömören összegzi a leckében található ismereteket. Mielőtt elolvasná az összeggést, a lényeg kiemelésével foglalja össze fejben a tanultakat! Ha valami nem jut eszébe, olvasson vissza bátran a tananyagban! Csak az önálló összefoglalás után vesse össze saját gondolatait a lecke összefoglalásával.

Az összeggést a frissen szerzett tudás ellenőrzésére használható **önellenőrző kérdések** követik. Soha ne mulassza el ezek áttekintését! Minden kérdéshez megtalálja a helyes választ is. Ezt lehetőleg ne olvassa el mindaddig, amíg önállóan nem sikerült felelnie a feltett kérdésre. A válaszok csupán arra valók, hogy ellenőrizze saját megoldása helyességét.

## **1.4 FORRÁSOK**

A tananyag elsajátításához különböző forrásokat, állományokat biztosítunk. Amennyiben a tananyag mellé lemezmellékletet kapott azon megtalálja a ezeket az fájlokat. Ha tananyagot elektronikus környezetben sajátítja el, a letölthető fájlok elérhetők a kurzus felületén.



## 2. LECKE: MYSQL-ADATBÁZISOK KEZELÉSE

### 2.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Most következő leckénket egyfajta alapvetésnek szánjuk, amelyben röviden áttekintjük a továbbhaladáshoz szükséges ismereteket.

Tananyagunk olyan sorozat része, amelynek előző eleme a relációs adatbázisok koncepcionális, logikai, valamint fizikai tervezésével foglalkozik. Bár az említett tananyag megismerteti az olvasót mindazokkal az ismeretekkel, amelyekre jelen tananyagunk épül, fontosnak tartjuk azokat röviden összefoglalni. Ebben a leckében azokat a fogalmakat és tudnivalókat tekintjük át, amelyek feltétlenül szükségesek a további fejezetek megértéséhez és feldolgozásához.

A lecke végére Ön képes lesz képes lesz kapcsolatot fölépíteni a MySQL-adatbázis-szerverekkel. Meg tudja változtatni a kliens karakter kódolását, szabályozni tudja jelszavát, ismerni fogja az SQL-mondatok egyenkénti, és köteget végrehajtásának módját.

A lecke olvasása közben keressen válaszokat a következő kérdésekre!

- Mit értünk az alatt, hogy az adatbázis-kezelés aszinkron kommunikáció?
- Milyen elemek alkotják az adatbázis-rendszert?
- Mi a feladata az adatbázis-kezelő rendszernek?
- Hogyan használhatjuk egy adatbázis-kezelő rendszer szolgáltatásait?
- Hogyan kapcsolódhatunk a MySQL adatbázis-kezelő rendszerhez?
- Hogyan adhatjuk ki parancsainkat?
- Hogyan menthetjük állományba lekérdezéseink eredményét?
- Milyen módszerrel változtathatjuk meg jelszavunkat?

### 2.2 ADAT, INFORMÁCIÓ KOMMUNIKÁCIÓ

Az informatikus számára a világ különböző **objektumok** halmaza. Ebben az értelemben objektum minden olyan dolog, ami más dolgoktól megkülönböztethető. Objektum lehet egy bizonyos élőlény, egy tárgy, fogalom, jelenség, vagy bármi más. Az objektumok megkülönböztetésére alkalmas jegyeket jellemzőknek, **tulajdonságoknak** nevezzük. A tulajdonságok leírhatók, lerajzolhatók, kimondhatók, lefényképezhetők, azaz ábrázolhatók.

**Adatról** akkor beszélünk, amikor egy objektum valamilyen tulajdonságát mások (ember, vagy számítógép) által értelmezhető formában ábrázoljuk. Az értelmezés azt az aktív tevékenységet jelenti, amellyel a befogadó „megfejt” egy adat, neki szóló jelentését. Ezt a jelentést nevezzük **információnak**.

Az adat és információ közötti fontos különbség, hogy míg az adat ábrázolható, azaz van fizikai reprezentációja, addig az információ csak az emberi tudatban létezik. Ebből következik az is, hogy még az adat objektív, addig az információ (mivel az egyén „tulajdona”) szubjektív.

Az életünk során rengeteg adatot értelmezünk, ennek megfelelően hatalmas tömegű információra teszünk szert. Ez az információ tömeg alkotja **tudásunkat**, azt a szellemi potenciált, ami fizikai adottságaink mellett másoktól megkülönböztet bennünket.

## 2.3 KOMMUNIKÁCIÓ

Az emberi létforma evolúciós rátermettségének és az ember által birtokolt globális tudás hihetetlen bővülésének egyik legfontosabb fundamentuma a kifinomult **kommunikáció**. Az a folyamat, aminek során az emberek közvetve ugyan, de információt osztanak meg egymás között.

A „közvetve” kitétel arra utal, hogy az információ közvetlen átadására, egyfajta gondolatátvitelre ma még nincs ismert lehetőség.

Ha információt akarunk átadni, akkor meg kell alkotni annak valamiféle anyagi reprezentációját. Le kell írunk, rajzolunk, ki kell mondanunk, le kell festenünk gondolatainkat. Mindezt persze olyan formában kell megtennünk, hogy a kommunikációs partner képes legyen azt értelmezni.

Az információ anyagi reprezentációjának megalkotását kódolásnak is nevezzük. A kódoláskor valójában olyan adatokat hozunk létre, amelyek értelmezésével, dekódolásával a kommunikációs partner az eredetihez hasonló információhoz juthat hozzá.

A kommunikáció egyszerűsített folyamata a következő:

Az adó megalkotja az általa birtokolt információ anyagi reprezentációját, azaz adattá kódolja az információt.

Az adatot valamilyen továbbítására alkalmas közegen (kommunikációs csatornán) keresztül eljuttatja a vevőnek.

A vevő értelmezi, dekódolja a kapott adato(ka)t, és információhoz jut.

A kommunikáció annál hatékonyabb, minél inkább hasonlít a vevőben kialakuló információ az adó eredeti információjához. A hatékonyságot számos

dolog befolyásolhatja. Ilyen a kódolás és dekódolás technikája, vagy a kommunikációs csatorna hatékonyságát zavaró zaj.

A kommunikációs folyamat fenti sémája számos különböző formában fordulhat elő. Most a kommunikáció egy számunkra fontos jellemzőjét megragadva, tegyünk különbséget **szinkron** és **aszinkron** kommunikáció között.

**Szinkron** kommunikációról beszélünk akkor, ha a kódolás és dekódolás közötti **időbeni különbség elhanyagolható**, és a kommunikációs szerepek folyamatosan fölcserélődhetnek. Ilyenkor a kommunikáló felek egyfajta párbeszédet folytathatnak. Szinkron kommunikáció például a hagyományos és a telefonbeszélgetés.

Az **aszinkron** kommunikáció során a kódolás és **dekódolás időpontja között lényeges különbség** van. A kódoláskor, és az adatok továbbításakor a vevő nem áll készen az adatok fogadására, ezért a dekódolásig eltelt idő alatt, valamilyen adattároló eszköz segítségével meg kell oldani az adatok tárolását. Ilyen kommunikáció például a hagyományos postai levelezés, vagy az SMS-küldés, sőt, a nyomtatott dokumentumok készítése is. Az első esetben a postaláda, a másodikban a telefonkészülék, a harmadikban a nyomtatott dokumentum az az eszköz, ami továbbított adatokat a dekódolásig tárolja.

## 2.4 ADATBÁZIS-RENDSZEREK

A kommunikáció során tehát adatokat osztunk meg azért, hogy partnereinkben információ alakulhasson ki.

Pontosan ezt tesszük az adatbázisok használata során is. Az adatbázis-kezelés valójában nem más, mint aszinkron kommunikáció:

- A pillanatnyilag rendelkezésünkre álló információt adattá kódoljuk.
- Jól meghatározott adatszerkezetben,
- informatikai eszközök igénybevételével tároljuk.
- Illetve az így tárolt adatokat, ismét csak informatikai eszközök segítségével kinyerjük, majd feldolgozzuk, értelmezzük.

Mint látjuk az adatbázis-kezelés fontos jellemzője az informatikai eszközök, más néven információfeldolgozó eszközök használata.

Az adatbázisok kezelését megvalósító informatikai rendszereket adatbázis-rendszereknek nevezzük.

Az adatbázis-rendszerek három különböző összetevőből épülnek fel:

- adatbázisok

- adatbázis-kezelő rendszerek
- adatbázis-alkalmazások.

### Adatbázis

Az adatbázis a valamilyen szempontból logikailag összetartozó adatok és azok metaadatainak halmaza, amelyet meghatározott adatmodell alapján kialakított adatszerkezetben tárolunk. Tananyagunkban a relációs adatmodell szerint szervezett adatbázisok kezelésével foglalkozunk.

### Adatbázis-kezelő rendszer

Az adatbázis-kezelő rendszer vagy DBMS (Database Management System) olyan szoftver, amely lehetővé teszi az adatbázisok, és az azokban tárolt adatok manipulálását.

Alapvető feladatai a következők:

- adatok fizikai kezelése,
- logikai adatmodell szerinti adatkezelés biztosítása,
- adatbázisok létrehozása,
- adatbázisok szerkezetének kialakítása,
- adatok tárolása,
- párhuzamos (egy időben több felhasználó általi) hozzáférés biztosítása,
- felhasználói jogosultságok ellenőrzése és kezelése,
- adatok kinyerésének, lekérdezésének, változtatásának biztosítása,
- adatintegritás megtartása,
- hálózati hozzáférés megvalósítása.

Az adatbázis-kezelő rendszerek általában hálózaton elérhető **kiszolgáló alkalmazásként**, azaz szerverként működnek. Általában nem rendelkeznek kezelőfelülettel, hanem a kliensalkalmazások segítségével küldött speciális **adatkezelő nyelven** leírt utasításokkal **vezérelhetők**. A **relációs adatbázis-kezelő rendszerek adatkezelő nyelve az SQL**.

Ezeket a kliensalkalmazásokat nevezzük **adatbázis-alkalmazásoknak**. Az adatbázis-alkalmazások felületet biztosítanak az adatbázis-kezelő rendszerhez való hozzáféréshez. Általában teljes mértékben eltakarják a felhasználó elől az adatbázis-kezelő műveleteket, így a felhasználónak nem is kell tudnia, hogy a képernyőn megjelenő, és kezelt adatok honnan származnak.

Az ilyen adatbázis alkalmazások esetén a felhasználónak nincs szüksége – sem lehetősége – arra, hogy parancsokat küldjön az adatbázis-kezelő rendszernek.

Az adatbázisok adminisztrálására, menedzselésére speciális adatbázis-alkalmazásokat, úgynevezett adatbázis-segédprogramokat használhatunk. Ezek lehetővé teszik, hogy a felhasználó kapcsolódjon az adatbázis-kezelő rendszerhez, és azt pontosan megfogalmazott adatkezelő parancsokkal vezérelje. Az adatbázis-segédprogramokat gyakran nevezzük egyszerűen **kliensnek**.

A legtöbb adatbázis-kezelő rendszerhez karakteres és grafikus felületet biztosító kliensek is beszerezhetők.

## 2.5 ADATMODELL

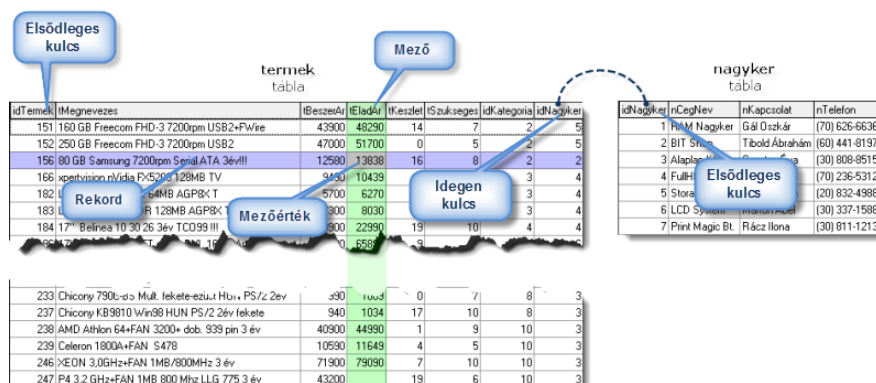
Az adatbázis-kezelés során objektumok, azaz egyedek tulajdonságait, és az egyedek közötti kapcsolatokat kezeljük.

Az azonos jellemzőket leíró tulajdonságokat tulajdonságtípusokba, az azonos tulajdonságtípusokkal rendelkező egyedeket pedig egyedtípusokba soroljuk.

**Az egyedtípusokba rendezett egyedeket, azok tulajdonságtípusokba sorolt tulajdonságait és az egyedek kapcsolatait mindig valamilyen adatmodell szabályai szerint kialakított adatszerkezetben tároljuk.**

Napjaink legelterjedtebb adatmodellje a relációs adatmodell, amely táblákban tárolja az egyes egyedtípusokba tartozó egyedek, tulajdonságtípusokba tartozó tulajdonságait.

Minden egyedtípus egy-egy **táblában** tárolódik. A tábla sorokból és oszlopokból álló mátrix, amelynek oszlopai az egyedtípus tulajdonságtípusait tartalmazó **mezők**, sorai az egyedeket tároló **rekordok**, a sorok és oszlopok találkozásánál pedig a tulajdonságokat tartalmazó **mezőértékek** találhatók. A tábla egyedeinek azonosíthatóságát az **elsődleges kulcs**, a különböző egyedtípusok egyedei közötti kapcsolatok tárolását pedig az **idegen kulcs** valósítja meg.



1. ábra Táblák a relációs adatmodellben

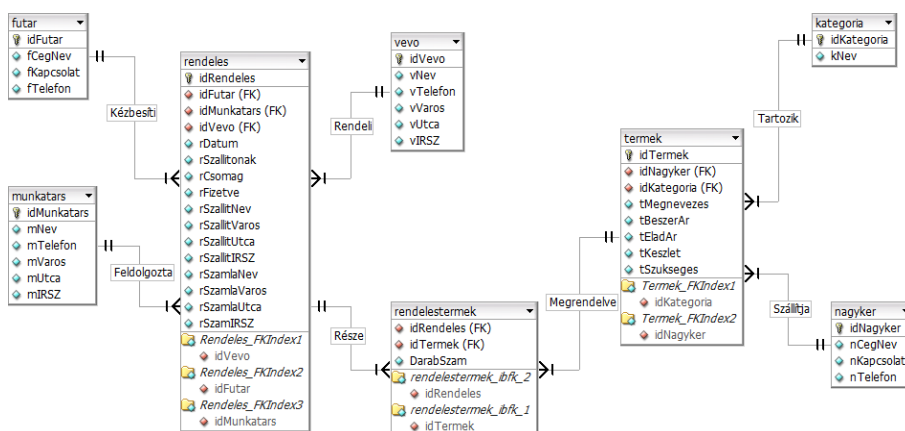
## 2.6 GYAKORLÓ ADATBÁZIS

Az adatbázis konkrét adathalmaz, amelyet egy adatmodell szabályainak betartásával alakítottak ki. Tananyagunkban a relációs adatmodell szabályait követő, azaz relációs adatbázisok kezelését fogjuk megismerni.

A tanultak megértéséhez és kipróbálásához egy informatikai eszközt forgalmazó – természetesen csak elképzelt – webes kereskedés adatbázisát használjuk majd. Az adatbázis erősen egyszerűsített, nem a valós felhasználást, hanem elsősorban a gyakorlást szolgálja. (Mindemellett kis átalakítással gyakorlati felhasználásra is alkalmassá tehető.)

A példában használt **webbolt** nagykereskedésekből (**nagyker** tábla) származó és különböző kategóriákba tartozó (**kategoria** tábla) termékeket (**termek**) forgalmaz.

A termékeket a **vevok** táblában tárolt vevők rendelhetik meg úgy, hogy a **rendeles** táblában létrehozott megrendelésekhez egy vagy több terméket kapcsolnak. Minden megrendelést a bolt egy-egy dolgozója (**munkatars**) kezel, és a céggel kapcsolatban álló futárszolgálat (**futar**) szállít ki.



2. ábra A webbolt adatbázis táblái

## 2.7 MYSQL ADATBÁZIS-KEZELŐ RENDSZER

Az adatbázis-rendszerek második eleme az adatbázis-kezelő rendszer. Tananyagunkban az ingyenesen használható DBMS-ek között, napjainkban talán legelterjedtebbnek nevezhető MySQL 5.5.0 verzióját, annak Windows operációs rendszeren futó változatát fogjuk használni.

A MySQL eredetileg a svéd MySQL AB cég terméke. SQL nyelven vezérelhető, hálózati környezetben kiszolgálóként működő, többfelhasználós relációs adatbázis-kezelő rendszer. A szoftver 2010 januárja óta az Oracle tulajdonában van. Kereskedelmi és GPL-licenccel is használható. Ingyenes használatának, illetve a PHP-beli integrációjának köszönhetően a webes alkalmazások támogatására talán legelterjedtebben használt adatbázis-kezelő rendszer.

Számos platformon, többek között Windows, MAC OS X, Linux, Solaris operációs rendszereken is használható.



<http://dev.mysql.com/downloads/mysql/>

1. link MySQL letöltése

A MySQL szerver telepítésére nem térünk ki. A szoftver, és dokumentációja letölthető a <http://dev.mysql.com/downloads/mysql/> címről.

Amennyiben az olvasó később adatbázis-háttérrel rendelkező webalkalmazások készítését tervezi, érdemes megfontolni valamilyen WAMP-csomag letöltését és telepítését.

A WAMP csomagok Windows platformon futó Apache webszervert, PHP-t, és MySQL-szervert telepítenek gépünkre. Ilyen csomag az úgynevezett EasyPHP, ami az alábbi címen férhető hozzá:

<http://www.easyphp.org/download.php>



<http://www.easyphp.org/download.php>

## *2. link EasyPHP letöltése*

### **2.7.1 Az SQL**

A relációs adatbázis-kezelő rendszerek napjainkra szabvánnyá vált nyelve az **SQL** (Structured Query Language), amely az IBM által, az 1970-es évek elején kifejlesztett SEQUEL (Structured English Query Language) utóda.

Az SQL-t 1986-ban az ANSI (Amerikai Nemzeti Szabványügyi Intézet – American National Standards Institute), később pedig az ISO is hivatalos szabvánnyá minősítette. Napjainkra szinte minden relációs adatbázis-kezelő szoftvert gyártó cég terméke, így értelemszerűen a MySQL is támogatja.

Az ANSI- és ISO-szabványok biztosítják a nyelv egységességét, a különböző gyártók implementációi, az úgynevezett SQL-nyelvjárások azonban mégis mutatnak kisebb-nagyobb eltéréseket. Tananyagunkban természetesen a MySQL adatbázis-kezelő rendszer vezérlésére alkalmas SQL változatot mutatjuk be.

Az SQL-ben úgynevezett SQL-mondatok formájában fogalmazhatunk meg egy-egy adatbázis kezelő parancsot. Az nyelv négy nagyobb résznelvből tevődik össze.



Az **SQL DDL** (Data Definition Language = Adatdefiníciós Nyelv) tartalmazza adatszerkezetek létrehozását biztosító nyelvi elemeket. A DDL segítségével új adatbázisokat, és táblákat hozhatunk létre, de módosítjuk, illetve törölhetjük is azokat.

A **DCL**-t (Data Control Language = Adatelérést Vezérlő Nyelv) résznyelvbe azokat nyelvi elemeket sorolják, amelyekkel az adatbázis-kezelő rendszer működését szabályozhatjuk.

**SQL DML** (Data Manipulation Language = Adatmanipulációs Nyelv) biztosítja az új rekordok beszúrásának, az adatok megváltoztatásának és törlésének lehetőségét. A DML teszi lehetővé, hogy az adatbázis tartalma bármikor változtatható legyen.

**SQL DQL** (Data Query Language = Adatlekérdező Nyelv) az SQL adatlekérdező résznyelve, amely az adatbázisokban tárolt adatok kiolvasását teszi lehetővé.

A DCL-t gyakran tekintik a DDL, a DQL-t pedig a DML részének, így egyes irodalmakban az SQL-t csak két résznyelvre DDL-re és DML-re osztják.

Még előző tananyagunkban elsősorban a DCL és DDL résznyelvekkel foglalkoztunk, addig jelen **tananyagunkban a DQL és DML résznyelvekre koncentráltunk**.

## 2.7.2 Az SQL nyelvtana

Az SQL-ben, a számítógép programozás során leginkább alkalmazott procedurális programozási nyelvekkel szemben nem a feladat megoldását kell megfogalmaznunk, hanem a várt eredményt kell meghatároznunk a nyelv eszközeinek felhasználásával. E tekintetben az SQL az úgynevezett **deklaratív** programozási nyelvekhez hasonló.

Az SQL mondatok mindig egy paranccsal kezdődnek, és pontosvesszővel (;) fejeződnek be.

A parancsot meghatározott sorrendben, különböző kötelező és opcionális elemek követhetik:

- az SQL fenntartott szavai,
- tábla-, mező- és változóazonosítók,
- értékek, literálok,
- kifejezések,
- függvényhívások.

### SQL-mondatok tördelése

Az SQL mondatok szóhatárokon tetszőleges számú sorba tördelhetők. A mondat végét nem a sortörés, hanem pontosvessző jelzi.

A mondatokat, egyenként vagy kötegelten is végrehatathatjuk a MySQL-szerverrel. Utóbbi esetben egy-, vagy többsoros megjegyzések is elhelyezhetők a szövegben. Az egy soros megjegyzések kettőskereszt (#) karakterrel kezdődnek, a többsoros megjegyzéseket pedig /\* \*/ jelek közé zárjuk.

### Kis- és nagybetűk

Az értelmező nem tesz különbséget a kis- és nagybetűk között, de az adatbázis önálló állományként tárolódó elemeinek azonosítói (például táblák) az operációs rendszertől függően kivételt jelenthetnek.

### Szövegek

A **szöveg literálok**at a MySQL-ben is egyszeres vagy dupla aposztrófok közé kell zárni.

### Szintaktikai leírás

Az SQL-mondatok szintaktikáját, az egyes parancsok specifikációját az alábbi formában fogjuk ismertetni (az itt olvasható elemek nem részei az SQL nyelvnek, csupán a szintaktikai leírás egyes részeinek használati módjára utalnak):

- NAGYBETÜKKEL írt szavak: leírásban megadott formában használható nyelvi elemek
- [szögletes zárójelben megadott szöveg]: opcionális nyelvi elem. A nem szögletes zárójelek között lévő nyelvi elemek használata kötelező.
- | függőleges vonal (cső karakter) két szöveg között: választási kötelezettség. A cső karakterrel elválasztott elemek közül kizárólag egy használható.
- Dőlt betűk: értelemszerűen helyettesítendő szöveg.
- Sortörés jel (↵): a csak több sorban elférő szintaktikai leírások logikai összekapcsolására használjuk!

Az SQL-mondatok és parancsok formai leírása általában több opcionális (elhagyható) elemet is tartalmaz. Az opcionális elemek használatát az határozza meg, hogy milyen eredményt szeretnénk kapni.

Fontos azonban tudnunk, hogy a leírásban feltüntetett elemek sorrendje általában kötött. A használt opciókat a specifikációnak megfelelő sorrendben kell szerepeltetnünk.

Az alábbi szintaktikai leírás a felhasználó jelszavának megváltoztatására alkalmas DCL-mondat általános formátumát határozza meg. A **[FOR user\_azonosító]** szögletes zárójele jelzi, hogy a közrezárt rész használata nem kötelező, a *user\_azonosító* szó dőlt betűje pedig arra utal, hogy a szó helyére egy felhasználó azonosítójának kell kerülnie. A harmadik, és negyedik sor elején lévő cső karakter jelzi, hogy az SQL-mondatban a 2., vagy a 3., vagy pedig a 4. sorban leírt részt kell használni.

```
SET PASSWORD [FOR user_azonosító] =  
    PASSWORD('jelszó')  
    | OLD_PASSWORD('jelszó')  
    | 'kódolt jelszó'
```

A fenti szintaktikai leírásnak megfelel például az alábbi SQL-mondat:

```
SET PASSWORD = PASSWORD('sosemtudodmeg')
```

## 2.8 KAPCSOLÓDÁS MYSQL-SZERVERHEZ

A modern adatbázis-kezelő rendszerekhez általában hálózati kapcsolaton keresztül, adatbázis-alkalmazások segítségével férünk hozzá. A tanultak kipróbálásakor, a MySQL szerver teljes körű vezérlését lehetővé tévő MySQL klienst használunk majd. A **mysql** nevű, karakteres vezérlést biztosító alkalmazás a szerverrel együtt telepíthető, éppen ezért ezt fogjuk majd alkalmazni.



<http://www.sqldeveloper.net/>

### 3. link DreamCoder letöltése

A tanultak természetesen grafikus felületű kliensekkel kipróbálhatók és végrehajthatók. Ilyen kliens például DreamCoder for MySql nevű grafikus al-

kalmazás, amelynek ingyenesen letölthető változatát megtaláljuk a <http://www.sqldeveloper.net/> címen.

### 2.8.1 A MySql parancsori kliense

A MySQL parancssori kliense, a **mysql** nevű alkalmazás, része az adatbázis-kezelő rendszer telepítő csomagjának. A kliens server telepítését követően, több másik segédprogrammal együtt, a **mysql** programkönyvtárának **bin** mappájában található meg.

Könyvünk példáiban a MySQL programkönyvtára a **C:** meghajtó **mysql** mappája.

#### Kliens indítása

A kliens a **mysql** paranccsal indítható, ami számos opcionális paraméterrel látható el. A paraméterek kötőjellel kezdő egybetűs (**-u**), vagy dupla kötőjellel kezdődő szó (**--help**) formátumúak lehetnek.

Ha a bővebb információt szeretnénk kapni róluk, a **mysql --help** parancs hatására megjelenő súgóban a legtöbb opciót megtaláljuk. Ha minden lehetséges beállítást látni akarunk, használjuk az alábbi módon a parancsot:

```
mysql --help --verbose
```

A server beállításaitól függően előfordulhat, hogy a kliens minden paraméter nélkül is elindul. Általában azonban meg kell adnunk az adminisztrátorától kapott felhasználói nevünket (**-u felhasználói\_név**) és ha van jelszavunk, jeleznünk kell a használatának szándékát is (**-p**).

Az alábbi példában adatbázis adminisztrátorként jelentkezünk be. A server telepítése után az adminisztrátornak nincs jelszava, ezért a **-p** kapcsolóra nincs szükség.

```
C:\mysql\bin>mysql -u root
```

A kliens indítása után (ha használtunk **-p** opciót) beolvassa a jelszót, majd felhasználói névvel együtt elküldi a servernek.

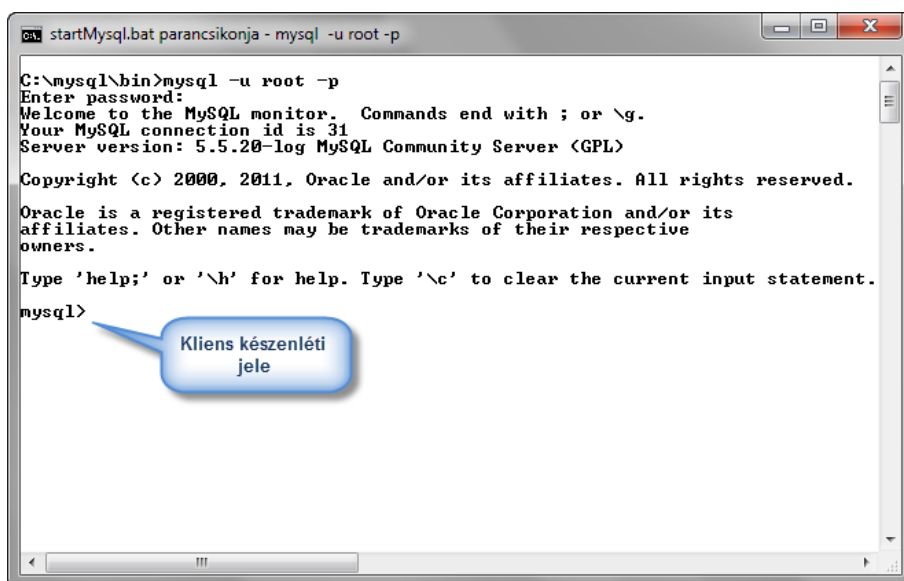
A server ellenőrzi a felhasználó-jelszó-host adathármashoz tartozó jogosultságokat, majd ezeknek megfelelően engedélyezi, vagy elutasítja a belépést.

A bejelentkezéshez használt fenti forma akkor működőképes, ha a MySQL szerver a saját gépünkön található, és a szokásos **3306**-os portot figyeli. Ellentérő esetben használnunk kell a **-h** *host\_cím*, illetve a **-P** *port* paramétereket.

Hasznos lehet a **-default-character-set=karakterkészlet** paraméter, amellyel jelezhetjük a szervernek, hogy milyen karakterkészletet használ a kliens. A karakterkészlet megadásakor a szerveren telepített karakterkészletek között választhatunk. Ha a kliens magyar nyelvű Windows operációs rendszer alatt fut, a **-default-character-set=cp852** a helyes beállítás.

```
C:\mysql\bin>mysql -u root -default-character-set=cp852
```

A sikeres kapcsolódás után megjelenik a kliens készenléti jele, ellenkező esetben hibaüzenetet kapunk.



3. ábra A kliens indulás után

A készenléti jel után már a szervernek szóló SQL-mondatokat gépelhetjük be, de néhány kapcsolóval a kliens működését is szabályozhatjuk. Ezek a kapcsolók és funkcióik rövid leírása a kliens felületén kiadott **help**, vagy **\h** utasításokkal jeleníthetők meg.

A készenléti jel után begépelte SQL-mondatok a szerverhez kerülnek, az eredmény pedig a kliens felületén jelenik meg.

A MySQL finoman hangolható jogosultsági rendszerrel rendelkezik, amit az arra kijelölt felhasználók szabályozhatnak.

Tananyagunkban nem térünk ki az egyes műveletekhez szükséges jogosultságokra. Abból indulunk ki, hogy a használt adatbázisban minden privilégiummal rendelkezik az olvasó.

A kapcsolat megszakításához, és kliens bezárásához a **quit** vagy a **\q** parancs használható.

```
mysql>quit  
C:\mysql\bin>
```

### Parancsok begépelése

Bejelentkezés után megjelenik kliens készenléti jele (**mysql>**) ami mellé rendre begépelhetjük pontosvesszővel záródó SQL-mondatainkat. A parancsok az ENTER leütése után kerülnek végrehajtásra. Az alábbi módon kiadott parancs hatására a kliens kilistázza azokat az adatbázisokat, amelyek eléréséhez a bejelentkezett felhasználó jogosultsággal rendelkezik.

```
mysql>SHOW DATABASES;
```

Jelszavunk beállítását szintén elvégezhetjük a kliens felületén:

```
SET PASSWORD=PASSWORD('új_jelszó');
```

Ha egy parancs elvégzése közben hiba keletkezik, a DBMS megszakítja a végrehajtást, majd visszaállítja az utasítás kiadása előtti állapotot. Egy parancs tehát vagy teljes egészében végrehajtásra kerül, vagy egyáltalán nem.

### Kötegetelt végrehajtás

A fenti technikával egyenként kell begépelnünk SQL-mondatainkat. Előfordulhat, hogy előre elkészített szövegfájlban tárolt mondatokat szeretnénk, kötegetelt módon végrehajtani. Ilyenkor a kliens felületén használjuk a **SOURCE** *fájlnev* parancsot, ahol a *fájlnev* a szövegfájl pontos elérési útja. Az alábbi parancs hatására a klienst végrehajtja az főkönyvtár **sqls** mappájának **webbolt.sql** nevű állományában található SQL-mondatokat.

```
mysql> SOURCE c:/sqls/webbolt.sql
```

Hasonló hatást érünk el, ha indításkor irányítjuk az SQL-mondatokat tartalmazó szövegfájl tartalmát a kliensbe. Ez utóbbi esetben azonban a parancsok végrehajtását követően, a kliens azonnal befejezi működését.

```
C:\mysql\bin>mysql -u root < c:\sqls\webbolt.sql
```

A **mysql** alkalmazással dolgozva az eredmény, mindig a kliens felületén jelenik meg. Lehetséges, hogy a megjelenített adatokat külső szövegfájlba szeretnénk irányítani. Ezt legegyszerűbben a kliens indítása előtt, a kimenet fájlba irányításával tehetjük meg. Az alábbi parancs az **c:\sqls** mappa **kimenet.txt** állományába küldi a kliens által megjelenített adatokat. A megoldás apró szépségbája, hogy sem a kimenet, sem pedig a kliens készenléti jele nem jelenik meg a képernyőn:

```
C:\mysql\bin>mysql -u root > c:\sqls\kimenet.txt
```

A kimenet fájlba irányításának szebb megoldása, amikor a kliens felületén a **tee** paranccsal (**\T**) adjuk meg a kimeneti fájlt. Ezt követően a készenléti jel, és minden parancsunk megjelenik a képernyőn, de egyben a kimenetként megadott fájlban is. Az ilyen naplózást a **notee**, vagy **\t** paranccsal kapcsolhatjuk ki.

Ha köteget végrehajtás közben hiba keletkezik, csak a hibát okozó SQL-mondat végrehajtása szakad meg. A fájl feldolgozása tovább folytatódik.

### Gyakorló adatbázis telepítése

A további példák a korábban említett **webbolt** adatbázisra épülnek. Az adatbázis létrehozásához le kell töltenie és a megfelelő mappába kell mentenie a **webbolt.sql** nevű állományt! A fájlt innen töltheti le:



1. letöltés *webbolt.sql*

A letöltést követően köteget módon végre kell hajtani fájlban található SQL-mondatokat.

A művelet sikeres végrehajtása után rendelkezésére fog állni a webbolt nevű adatbázis, amelyben minden további feladatot, és gyakorlatot elvégezhet.

```
C:\mysql\bin>mysql -u root < c:\sqls\webbolt.sql
```

A fenti művelet akkor lesz sikeres, ha adminisztrátorként fér hozzá az adatbázis-szerverhez. Ha ez nem így van, kérje meg az adminisztrátort az adatbázis telepítésére, majd az eléréshez szükséges teljes jogosultság biztosítására.

Az adatbázis létrehozása után célszerű létrehozni a szerveren egy felhasználót, akinek nevében használni fogja a gyakorló adatbázist. Ehhez a kliens indítása után gépelje be a következő parancsot:

```
GRANT ALL ON webbolt.* TO webadmin@localhost;
```

A továbbiakban **webadmin** névvel éri el az adatbázist, és természetesen be is jelentkezhet a felhasználó névvel:

```
mysql -u webadmin -p --default-character-set=cp852
```

### Adatbázis kiválasztása

A kliens felületére bejelentkezve bármelyik adatbázist használhatjuk, amelyhez megfelelő jogosultsággal rendelkezünk. Éppen ezért, mielőtt valamilyen egyéb parancsot kiadnánk, célszerű kiválasztani azt az adatbázist, amellyel éppen dolgozni szeretnénk. Erre a **USE** parancsot használhatjuk:

```
USE adatbázis_neve;
```

A **webbolt** adatbázis használatbavétele így történik:

```
USE webbolt;
```

## 2.9 MUNKAMENET BEÁLLÍTÁSA

Amikor a felhasználó a saját gépen futó klienssel bejelentkezik az adatbázis-kezelő rendszerbe, a szerver és a kliens között kapcsolat alakul ki, amit **munkamenetnek**, vagy **session**-nek nevezünk. A session mindaddig tart, amíg a



felhasználó ki nem jelentkezik, vagy a kapcsolat valamilyen hálózati hiba miatt nem megszakad.

A MySQL-szerver működése számos, névvel azonosítható jellemző, úgynevezett **rendszerváltozó** értékével befolyásolható. A rendszerváltozók egy része **statikus**, a szerver működése közben nem változtatható, de más részük **dinamikus**, menet közben is szabályozható. A változók **globális**- és **munkamenet** változók lehetnek. A globális változók a szerver működését befolyásolják, és a csak a DBMS adminisztrátora szabályozhatja őket, a session változók azonban a felhasználó aktuális kapcsolatára vonatkoznak, és a **felhasználó** maga **szabályozhatja** értéküket. A globális és session-változók értékének beállítása egyaránt az alábbi formában történhet:

```
SET [SESSION|GLOBAL] változónév=érték;
```

A változók aktuális értékének kiírását következőképpen kérhetjük. Ha a változó nevébe a % helyettesítő karaktert is elhelyezzük, akkor egyszerre több változó érték is kiírható:

```
SHOW VARIABLES LIKE 'változónév';
```

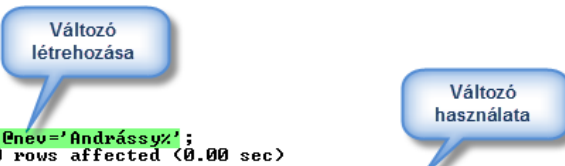
```
mysql> SHOW VARIABLES LIKE 'version%';
```

```
+-----+-----+
---+
| Variable_name          | Value
|
+-----+-----+
---+
| version                | 5.5.20-log
|
| version_comment        | MySQL Community Server
(GPL) |
| version_compile_machine | x86
|
| version_compile_os      | Win32
|
+-----+-----+
---+
```

A session-változók egyik típusát képezik a felhasználói változók. A felhasználói változók értékei nem szabályozzák a szerver, vagy a kliens működését, azonban lehetőséget adnak a felhasználó számára adatok átmeneti tárolására. A felhasználói változókat elsősorban SQL-mondatok eredményének tárolására és más mondatokban történő felhasználására alkalmazhatjuk.

Az ilyen változókat formailag, a nevük elé írt @ jel különbözteti meg a rendszerváltozóktól. Értékadásuk azonos módon történik, az érték kiírásához azonban a **SELECT** @változnév; parancsot kell használnunk.

Az alábbi példában (tananyagunkban kissé előreszaladva) a @nev változó létrehozását, értékének beállítását, majd a változó értéknek megfelelő nevű vevők lekérdezését mutatjuk be:



```
mysql> set @nev='Andrássy';
Query OK, 0 rows affected (0.00 sec)

mysql> select vnev from vev where vnev like @nev;
+-----+
| vnev |
+-----+
| Andrássy Mária |
| Andrássy Beatrix |
| Andrássy Hermína |
+-----+
3 rows in set (0.00 sec)
```

4. ábra Felhasználói változó használata

## 2.10 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 2.10.1 Összefoglalás

Ebben a leckében összefoglaltuk az adatbázis-kezelés tananyagunk elsajátításához szükséges alapfogalmait. Elsőként az adatbázisokból, adatbázis-kezelő rendszerből, és adatbázis-alkalmazásokból fölépülő adatbázis-rendszerről esett szó. Megtanultuk, hogy az adatbázisokkal kapcsolatos műveleteket az adatbázis-kezelő rendszer biztosítja, de a legtöbb DBMS csak adatbázis alkalmazáson keresztül, valamilyen adatkezelő nyelv segítségével vezérelhető.

Tananyagunk további részében a relációs adatbázisok kezelésére alkalmas, ingyenesen használható MySQL adatbázis-kezelő rendszert fogjuk használni, amelyet a DBMS telepítő csomagjának részeként beszerezhető **mysql** nevű adatbázis alkalmazás segítségével fogunk vezérelni.

A MySQL a mára szabvánnyá vált SQL nyelv segítségével vezérelhető. Az SQL négy résznyelvre (DCL, DDL, DQL, DML) bontható, melyek közül ebben a tananyagban, az adatok lekérdezésére alkalmas DQL, és az adatok változtatására használható DML résznyelveket tárgyaljuk részletesebben.

Az SQL nyelvvel írt SQL-mondatokban nem a feladat megoldásának módját, hanem a várt eredményt kell pontosan megfogalmazni. Minden SQL-mondat egy-egy adatkezelő művelet végrehajtására utasítja a DBMS-t. A mondatok többsorosak is lehetnek, zárásukat pontosvessző jelzi. A kliens felületén begépett mondatok azonnal végrehajtnak, de lehetőségünk van a szövegfájlokban tárolt SQL-parancsok kötegelt végrehajtására is.

Tananyagunk további részében egy képzeletbeli webes kereskedés, előre elkészített, **webbolt** nevű adatbázisát, ennek kezeléséhez pedig a **webadmin** felhasználói nevet fogjuk használni.

Az adatbázis a **webbolt.sql** kötegelt SQL-állomány letöltése és a **C:\sqls** mappába mentése után, az alábbi módon hozható létre:

```
C:\mysql\bin>mysql -u root < c:\sqls\webbolt.sql
```

A **webadmin** felhasználó jogosultságait a kliens felületén kiadott parancssal állíthatjuk be:

```
GRANT ALL ON webbolt.* TO webadmin@localhost  
IDENTIFIED BY 'jelszó';
```

## 2.10.2 Önellenőrző kérdések

1. Tesz-e a különbséget a MySql szerver a kis- és nagybetűk között?
  - Nem, de kis- nagybetű érzékeny operációs rendszerek esetén, a fájlként tárolódó elemek (pl. táblák) neveiben igen.
2. Lehet-e több SQL-mondatot kötegelt módon végrehajtani?
  - Igen. Ehhez az SQL-mondatokat szövegfájlban kell tárolni, majd a szövegfájl átirányításával kell elindítani a klienst. Hasonló eredményre vezet, ha a kliensben a **SOURCE** szót követően megadjuk a szövegfájl pontos elérését.
3. Azt tapasztalja, hogy az adatbázisban tárolt adatok ékezetes betűi helytelenül jelennek meg a kliensben. Mi a teendő?
  - A problémát a kliens karakterkódolásának hibás beállítása okozza. A kódolást indításkor a **--default-character-set=kódtábla** paraméterrel állíthatjuk be.
4. Kötelező-e egy sorban feltüntetni egy SQL-mondat teljes szövegét?

- Nem. Az SQL-mondatokat tetszőleges számú sorra törölhetjük. A mondat végét ; jelzi. Fontos azonban, hogy sortörést csak szóhatáron alkalmazzunk!

5. Hogyan állíthat be új jelszót?

- A kliens felületén az alábbi paranccsal:  
`set password=password('új_jelszó');`

## 3. LECKE: VÁLASZTÓ LEKÉRDEZÉSEK KÉSZÍTÉSE

### 3.1 CÉLKITÚZÉSEK ÉS KOMPETENCIÁK

Előző leckénkben az adatbázis-kezelést az aszinkron kommunikációhoz hasonlítottuk. Ebben a kommunikációs formában az adó kódolása nyomán létrejött adatokat valamilyen eszközzel tároljuk, hogy a vevő egy későbbi időpontban dekódolhassa, értelmezhesse azokat. Ezzel a módszerrel a vevő akkor juthat információhoz, amikor arra szüksége van.

Mostani leckénkben azokat az SQL-elemeket és módszereket fogjuk megismerni, amelyek segítségével az értelmezés érdekében kinyerhetünk az adatbázisból az abban tárolt adatokat.

Az adatok kinyerésére használt SQL-mondatokat választó lekérdezéseknek nevezzük, mert segítségükkel egy rekordforrásból (egy, vagy több tábla) rekordokat válogathatunk ki, és jeleníthetünk meg.

A lekérdezés megnevezés azért találó, mert ezekkel az SQL-mondatokkal az a **rekordforrásra** vonatkozó kérdéseket fogalmazhatunk meg, amelyeket a DBMS kiértékel, majd rekordok felsorolásával, az úgynevezett **eredményhalmazzal** válaszol.

Egyelőre a lekérdezések legalapvetőbb formájával, a DQL résznyelvbe tartozó SELECT parancs használatával foglalkozunk. Bár ebben a leckében csupán a parancs legegyszerűbb használatát tanuljuk meg (egyszerre egyetlen tábla rekordjaiból válogatunk), a későbbiekben látni fogjuk, hogy SELECT lehetőségei jóval szélesebbek. Az egyébként roppant egyszerű szintaktikájú parancs számos, bámulatosán érdekes lehetőséget biztosít. A későbbi leckékben a SELECT-tel készíthető választó lekérdezéseket több tábla kapcsolódó rekordjainak lekérdezésére, és különböző statisztikai műveletek lebonyolítására is használjuk majd. Szintén a későbbi leckék során térünk rá arra, hogy hogyan lehet több választó lekérdezés kombinációjával (úgynevezett allekérdezések készítésével) egészen megdöbbentő kérdéseket megfogalmazni, illetve, hogyan lehet egy választó lekérdezés eredményhalmazát DML-mondatoknak átadni.

A leckében keressen válaszokat az alábbi kérdésekre:

- Hogyan lehet megadni a lekérdezés rekordforrását, azaz azt a táblát, amelynek rekordjait látni szeretnénk?

- Hogyan szabályozható az eredményhalmazban megjelenő mezők száma és sorrendje?
- Hogyan lehet a táblában tárolt adatokkal számításokat végezni, és azok eredményét megjeleníteni?
- Hogyan lehet korlátozni az eredményhalmazban megjelenő rekordok számát, illetve tartalmát?
- Milyen módszerrel rendezhetők az eredményhalmaz rekordjai?

## 3.2 A SELECT PARANCSSAL ÁLTALÁNOS FORMÁJA

A **SELECT** paranccsal létrehozható választó lekérdezések hatására a DBMS egy rekordforrásból rekordokat és mezőket választ ki, majd azokat úgynevezett eredményhalmaz formájában visszaküldi a kliensek. Az eredményhalmaz megjelenítéséről a kliens gondoskodik.

A rekordforrás lehet egyetlen tábla, vagy több tábla kapcsolata. A rekordforrás rekordjai (egy tábla esetén annak összes rekordja, több tábla esetén azok kapcsolódó rekordjai) alkotják a forráshalmazt.

Az eredményhalmaz, a forráshalmaz lekérdezéssel kiválasztott rekordjaiból és mezőiből létrehozott mátrix. Szerkezetét tekintve éppn olyan, mint egy tábla, amely azonban sehol sem tárolódik, csupán a kliens felületén jelenik meg.

Ha meg kell mutatnunk az egri vevők nevét és telefonszámát, akkor a vevo tábla egyedül alkotja rekordforrást. Ennek megfelelően a forráshalmaz a tábla összes rekordja. Az eredményhalmaz az ábrán jelölt mezők és rekordok metszete lesz.

idUveo	vNev	vTelefon	vUares	vUtca	vIRSZ	vMail
1	Andrássy Mária	(680) 832-4378	Budapest	Kassa utca , 94	1811	andraszy.maria@gabriel.hu
20	Ratkó Öszkár	(680) 434-3282	Budapest	Halászcserda utca , 4	1811	ratko.oszkar@gabriel.hu
31	Uarga Jolán	(380) 451-2314	Eger	Kiskapu u. 14.	3300	NULL
17	Moka Ibolya	(780) 542-5872	Eger	Kovács Gy. utca , 69	3300	moka.ibolya@gabriel.hu
19	Kovács Lehel	(780) 989-4964	Eger	Thorma János utca , 13	3300	kovacs.lehel@quickpost.com
22	Kolonicz Zsuzsanna	(680) 980-5824	Eger	Szilvás köz utca , 9	3300	kolonicz.zsuzsanna@egorsgalamb
24	Berez Rebeka	(780) 825-3293	Eger	Péterka József utca , 8	3300	berez.rebeka@gmail.hu
25	Simon Lőrinc	(280) 362-7654	Eger	Kolmann utca , 95	3300	simon.lorinc@gmail.hu
28	Virga Izabella	(380) 319-2384	Eger	Uac utca , 74	3300	virga.izsola@gmail.hu
29	Keller Berta	(780) 626-6488	Eger	Kolmann utca , 47	3300	keller.bertha@egorsgalamb.hu
14	Kállai Gáspár	(280) 471-5761	Eger	Latabár utca , 1	3300	kallai.gaspar@gabriel.hu
6	Károly Jácint	(780) 264-7874	Eger	Szalag utca , 12	3300	karoly.jacint@gmail.hu
3	Galamb Huba	(680) 235-5859	Eger	Kolmann utca , 55	3300	galamb.huba@gabriel.hu
4	Károly Bertalan	(380) 891-6789	Eger	Cseresznye utca , 74	3300	karoly.bertalan@egpost.hu
10	Liktor Bertalan	(280) 984-2221	Eger	Platán utca , 17	3300	liktor.bertalan@gmail.hu
11	Lénárd Domokos	(680) 638-9383	Eger	Kard utca , 82	3300	lenard.domokos@egpost.hu
23	Andrássy Beatrix	(380) 747-7979	Gyöngyös	Szentgál utca , 21	3200	andraszy.beatrix@quickpost.com
2	Dallos Rebeka	(380) 584-4836	Gyöngyös	Mázsa utca , 5	3300	dallos.rebeka@gmail.hu
30	Hanga Berta	(380) 574-7970	Kazincbarcika	Kassa utca , 71	3700	hanga.bertha@gabriel.hu
8	Devényi Jácint	(280) 696-4781	Kazincbarcika	Henyer utca , 8	3700	devenyi.jacint@quickpost.com
7	Ponogács Eged	(680) 768-2916	Miskolc	Szilvás köz utca , 58	3501	ponogacs.eged@gabriel.hu
13	Pintye Lajos	(280) 737-7775	Miskolc	Szilvás köz utca , 38	3501	pintye.lajos@quickpost.com
9	Alapi Domokos	(380) 583-7765	Miskolc	Kovács Pál utca , 69	3501	alapi.domokos@gmail.hu
5	Keller Berta	(280) 942-4648	Miskolc	Egrikereszt utca , 33	3501	keller.bertha@bansver.com
21	Miklósi Emőd	(780) 378-6676	Miskolc	Úrút Szabó István utca , 24	3501	niklosi.emod@gabriel.hu
12	Szél Domokos	(780) 525-8777	Ózd	Kovács Pál utca , 4	3600	szei.domokos@gmail.hu
16	Dirko Rebeka	(680) 634-8517	Ózd	Bajnár utca , 97	3600	dirko.rebeka@egpost.hu
18	Pelle Boglárka	(380) 695-5918	Pétervására	Dékán Á. utca , 97	3250	pelle.boglarka@bansver.com
15	Szél Erika	(680) 756-3432	Pétervására	Uenyige utca , 10	3250	szei.erika@gabriel.hu
26	Birtalan Beatrix	(380) 249-5345	Salgótarján	Henyer utca , 61	3100	birtalan.beatrix@gabriel.hu
27	Andrássy Hermina	(680) 578-2956	Salgótarján	Híd utca , 18	3100	andraszy.hermina@gmail.hu

5. ábra Egy táblából álló rekordforrás

vNev	vTelefon
Galamb Huba	<60> 235-5859
Károly Bertalan	<30> 891-6709
Károly Jácint	<70> 264-7074
Liktor Bertalan	<20> 304-2231
Lénárd Domokos	<60> 638-9383
Kállai Gáspár	<20> 471-5961
Moka Ibolya	<70> 542-6072
Kovács Lehel	<70> 909-4964
Kolonics Zsuzsanna	<60> 900-5824
Bereg Rebeka	<70> 825-3293
Simon Lőrinc	<20> 362-9654
Uirga Izsó	<30> 339-2384
Keller Berta	<70> 626-6488
Uarga Jolán	<30> 451-2314

6. ábra A lekérdezés eredményhalmaza

**A SELECT általános formája:**

A **SELECT** parancs, bár alább látható általános formája első pillantásra bonyolultnak tűnhet, valójában meglehetősen egyszerű.

```

SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  oszlopmeghatározás [, oszlopmeghatározás...]
  [FROM rekordforrás
  [WHERE logikai_kifejezés]
  [ORDER BY mező | kifejezés| mező_pozíció
    [ASC | DESC], ...]
  [LIMIT {[eltolás,] sorok | sorok OFFSET eltolás}]]

```

Egyszerűsége abban áll, hogy a parancsot követő számos nyelvi elem jó része szögletes zárójelben van, azaz opcionális, elhagyható. Szinte az összes opciót mellőzhetjük, egyedül a **SELECT**-et követő mező, vagy mező felsorolás kötelező.

A szükséges nyelvi elemeket az határozza meg, hogy miféle eredményt szeretnénk látni az eredményhalmazban.

### 3.3 PÉLDÁK A SELECT PARANCSS ALKALMAZÁSAIRA

A **SELECT** mellett leggyakrabban használt nyelvi elemek (lásd a fenti példát) lehetőséget adnak a rekordforrás meghatározására, az eredményhalmazban

megjelenítendő mezők, rekordok kiválasztására, és sorba rendezésére. Leckénk következő szakaszai az alkalmazás egyes eseteit és az ilyenkor használandó nyelvi elemeket mutatják be.

### 3.3.1 SELECT használata értékek kiírására

A SELECT után kötelező legalább egy oszlopmeghatározás, vagy vesszővel elválasztott oszlopmeghatározásokból álló lista feltüntetése. Ezzel jelezzük, hogy milyen oszlopokat szeretnénk látni az eredményhalmazban.

Egy oszlopmeghatározás lehet literál (érték), kifejezés, vagy mezőhivatkozás.

Bár az e fajta használat nem jellemző, elvileg lehetséges, hogy az SELECT parancsot csak oszlop meghatározások követik, a mondat más elemet (pl. a rekordforrás megjelölését) nem is tartalmaz. Az ilyenkor használt oszlop meghatározások csak literálok, és kifejezések lehetnek.

Ilyenkor a DBMS egyetlen sorból álló eredményhalmazt állít elő, amelyben az oszlopnevek az oszlop meghatározások szövegei lesznek. Az egyetlen rekordban, a literálok helyén értékük, a kifejezések helyén pedig azok eredménye jelenik meg:

```
mysql> SELECT 'kettő', '+', 'kettő=', 2+2;
+-----+-----+-----+-----+
| kettő | + | kettő= | 2+2 |
+-----+-----+-----+-----+
| kettő | + | kettő= |    4 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

A fenti példában a 'kettő', a '+', a 'kettő=' egyszerű szöveg literálok, a 2+2 pedig kifejezés. Az eredményhalmazban felül az oszlopnevek (mező), alattuk pedig értékeik (mezőérték) jelennek meg.

### 3.3.2 Rekordforrás és mezők megadása

A fenti példánál jóval gyakoribb, hogy az eredményhalmazban egy kiválasztott tábla rekordjait, illetve az azokban tárolt mezőértékeket szeretnénk megjeleníteni. Ebben az esetben pontosan meg kell adnunk, hogy melyik rekordforrás (tábla) melyik mezőit szeretnénk látni. A mezők kijelölésére a SELECT-et követő oszlop meghatározások, a rekordforrás megadására pedig a FROM kulcsszó ad lehetőséget.



A mezőket az oszlop meghatározásokba írt hivatkozások, azaz mezőnevek, a rekordforrást a FROM kulcsszót követő táblahivatkozás azonosítja. (A FROM, valamint a később szereplő WHERE, ORDER BY és LIMIT kulcsszavakat gyakran nevezik a SELECT parancs záradékainak is, a SELECT-et követő oszlop meghatározások helyett pedig szokás mezőlistát említeni.)

A FROM záradék használatakor a DBMS már valódi lekérdezést végez. A rekordforrásként megadott tábla rekordjait forráshalmazként használva kiválogatja a SELECT után megadott mezőket, és visszaadja az eredményhalmazt. A mezők, a mezőlistában megadott sorrendben jelennek meg az eredményhalmazban.

Az alábbi példa a **vevo** tábla két mezőjét, a vevő nevét tartalmazó **vNev**, és a vevő telefonszámát tároló **vTelefon** mezőket jeleníti meg. Mivel más utasítást nem adtunk, az eredményhalmaz a forráshalmaz minden rekordját tartalmazza majd.

```
mysql> SELECT vNev,vTelefon FROM vevo;
```

vNev	vTelefon
Andrássy Mária	(60) 832-4378
Dallos Rebeka	(30) 584-4036
Galamb Huba	(60) 235-5859
Károly Bertalan	(30) 891-6709
Keller Berta	(20) 942-4648
Károly Jácint	(70) 264-7074
Pomogács Egyed	(60) 968-2916
Dévényi Jácint	(20) 696-4781
Alapi Domokos	(30) 583-3755
Liktor Bertalan	(20) 304-2231
Lénárd Domokos	(60) 638-9383
Szél Domokos	(70) 525-8777
Pintye Lajos	(20) 737-7775
Kállai Gáspár	(20) 471-5961
Szél Erika	(60) 756-3432
Dirkó Rebeka	(60) 634-8517
Moka Ibolya	(70) 542-6072
Pelle Boglárka	(30) 695-5918
Kovács Lehel	(70) 909-4964
Ratkó Oszkár	(60) 434-3202
Miklósi Emőd	(70) 378-6676
Kolonics Zsuzsanna	(60) 900-5824
Andrássy Beatrix	(30) 747-7979
Bereg Rebeka	(70) 825-3293

```
| Simon Lőrinc      | (20) 362-9654 |
| Birtalan Beatrix | (30) 249-8345 |
| Andrásy Hermina  | (60) 570-2956 |
| Virga Izsó       | (30) 339-2384 |
| Keller Judit     | (70) 626-6488 |
| Hanga Berta      | (30) 574-7970 |
+-----+
30 rows in set (0.00 sec)
```

A **vevo** tábla a most látott kettőn kívül több mezőt is tartalmaz. Ha mindet látni szeretnénk, a mezőlista helyén egy csillagot kell szerepeltetnünk (\*)! A csillag speciális mezőhivatkozás: az összes mezőt jelenti egyben. A következő példában a rekordszám nem változik, de forráshalmaz minden mezője megjelenik.

```
mysql> SELECT * FROM vevo;
+----+-----+-----+-----+-----+-----+
| idVevo | vNev          | vTelefon | vVaros | vUtca          | vIRSZ |
+----+-----+-----+-----+-----+-----+
| 1 | Andrásy Mária | (60) 832-4378 | Budapest | Kassa utca , 94 | 1011 |
| 2 | Dallos Rebeka | (30) 584-4036 | Gyöngyös | Máza utca , 5 | 3200 |
| 3 | Galamb Huba   | (60) 235-5859 | Eger      | Kolmann utca , 55 | 3300 |
| 4 | Károly Bertalan | (30) 891-6709 | Eger      | Cseresznyés utca , 74 | 3300 |
| 5 | Keller Berta  | (20) 942-4648 | Miskolc   | Epreskert utca , 33 | 3501 |
| 6 | Károly Jácint  | (70) 264-7074 | Eger      | Szalag utca , 12 | 3300 |
| 7 | Pomogács Egyed | (60) 968-2916 | Miskolc   | Szilvás köz utca , 58 | 3501 |
| 8 | Dévényi Jácint | (20) 696-4781 | Kazincbarcika | Henger utca , 8 | 3700 |
| 9 | Álapi Domokos | (30) 583-3755 | Miskolc   | Kovács Pál utca , 69 | 3501 |
| 10 | Lektor Bertalan | (20) 304-2231 | Eger      | Platán utca , 17 | 3300 |
| 11 | Lénárd Domokos | (60) 638-9383 | Eger      | Kard utca , 82 | 3300 |
| 12 | Szél Domokos  | (70) 525-8777 | Ózd       | Kovács Pál utca , 4 | 3600 |
| 13 | Pinter Lajos  | (20) 737-7775 | Miskolc   | Szilvás köz utca , 30 | 3501 |
| 14 | Kállai Gáspár | (20) 471-5961 | Eger      | Latahar utca , 1 | 3300 |
| 15 | Szél Erika     | (60) 756-3432 | Pétervására | Uenyige utca , 10 | 3250 |
| 16 | Dirko Rebeka  | (60) 634-8517 | Ózd       | Bajnár utca , 97 | 3600 |
| 17 | Moka Ibolya   | (70) 542-6072 | Eger      | Révész Gy. utca , 69 | 3300 |
| 18 | Pelle Boglárka | (30) 695-5918 | Pétervására | Dékányi Á. utca , 97 | 3250 |
| 19 | Kovács Lehel  | (70) 909-4964 | Eger      | Thorma János utca , 13 | 3300 |
| 20 | Ratkó Oszkár  | (60) 434-3282 | Budapest  | Halászszerda utca , 4 | 1011 |
| 21 | Miklósi Emőd  | (70) 378-6676 | Miskolc   | Uári Szabó István utca , 24 | 3501 |
| 22 | Kolonics Zsuzsanna | (60) 900-5824 | Eger      | Szilvás köz utca , 9 | 3300 |
| 23 | Andrásy Beatrix | (30) 747-7979 | Gyöngyös  | Szentgál utca , 21 | 3200 |
| 24 | Bereg Rebeka  | (70) 825-3273 | Eger      | Peterka József utca , 8 | 3300 |
| 25 | Simon Lőrinc  | (20) 362-9654 | Eger      | Kolmann utca , 95 | 3300 |
| 26 | Birtalan Beatrix | (30) 249-8345 | Salgótarján | Henger utca , 61 | 3100 |
| 27 | Andrásy Hermina | (60) 570-2956 | Salgótarján | Híd utca , 10 | 3100 |
| 28 | Virga Izsó     | (30) 339-2384 | Eger      | Vas utca , 74 | 3300 |
| 29 | Keller Judit   | (70) 626-6488 | Eger      | Kolmann utca , 47 | 3300 |
| 30 | Hanga Berta    | (30) 574-7970 | Kazincbarcika | Kassa utca , 71 | 3700 |
+----+-----+-----+-----+-----+-----+
30 rows in set (0.00 sec)
```

7. ábra Minden mező megjelenítése

### 3.3.3 Álnevek az eredményhalmazban

A fenti példákban látjuk, hogy az eredményhalmaz mezőnevei meggyeznek a tábla mezőneveivel. Ha ezeket túl hosszúnak, vagy nem elég beszédesnek találjuk, lehetőségünk van úgynevezett **álnevekkel** helyettesíteni őket. Ehhez csak annyit kell tennünk, hogy feltüntetjük a választott álnevet a SELECT utáni mezőlista megfelelő mezőneve után. Az eredményhalmaz megfelelő oszlopának elnevezése ezután nem a mező neve, hanem az álnev lesz.

```
mysql> SELECT vNev, Uásárló FROM vevő;
```

Uásárló
Andráss Mária
Dallos Beke
Gala
Kár
Ke
Kár
Mogács Gyed
Bereg Beke
Simon Lőrinc
Birtalan Beatrix
Andrássy Hermína
Virga Izsó
Keller Judit
Hanga Berta

30 rows in set (0.00 sec)

mysql>

8. ábra Álnév használata

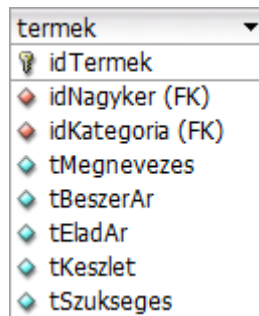
Álneveket több mezőnél is megadhatunk, sőt, arra is van lehetőség, hogy a táblák számára válasszunk alternatív nevet. (Erről a következő leckeiben olvashatunk.)

### 3.3.4 Számított mezőértékek

Előfordul, hogy olyan értéket szeretnénk megjeleníteni egy lekérdezés eredményhalmazában, amely nem szerepel az adatbázisban, de a tárolt mezők értékeiből kiszámolható.

- ☞ **Az forráshalmaz mezőinek értékeiből kiszámolt, de az adatbázisban nem tárolódó értékeket számított mezőknek nevezzük.**

Minta adatbázisunk **termek** táblája a web bolt által forgalmazott termékek adatait, többek között nevüket (**tMegnevezes**), beszerzési (**tBeszerAr**), és eladási árukat (**tEladAr**) tartalmazza. Nem szerepel azonban a táblában a két érték közötti különbség (haszon), és azok hányadosa (haszonkulcs).



9. ábra A termék tábla mezői

Láttuk, hogy amikor kifejezést adtunk meg a SELECT-et követő oszlop meghatározásokban, akkor az eredményhalmazban annak eredménye jelent meg ( $2+2 \Rightarrow 4$ ). A mezőlistában nyugodtan keverhetjük két módszert (mezők és kifejezések felsorolása), a kifejezésekben pedig mezőhivatkozásokat is használhatunk tényezőként.

Az adatbázis-kezelő rendszer minden visszaadott rekordban kiszámolja a kifejezések eredményét, és szerepelteti is az eredményhalmazban.

Számított mezőértékek alkalmazásakor általában álnevet rendelünk a számításhoz, különben az eredményhalmazban maga a kifejezés jelenik meg a számítás mezőneveként.

Az alábbi lekérdezés mezőlistájában a **tMegnevezes** mezőhivatkozás és két számított mező szerepel. Az egyik a **Haszon**, a másik a **Kulcs** álnevet kapta.

```
SELECT
    tMegnevezes, tEladAr-tBeszerAr Haszon,
    1-(tBeszerAr/tEladAr) Kulcs
FROM termék;
```

A lekérdezés eredményhalmazát a következő ábra mutatja.

```
mysql> select tMegnevezes, tEladár-tBeszerár Haszon, 1-(tBeszerár/tEladár) Kulcs FROM ternek;
```

tMegnevezes	Haszon	Kulcs
32MB/32bit EDO	129	0.0909
64MB/32bit EDO	219	0.0909
128 MB/64bit EDO	319	0.0909
256 MB PC3200	379	0.0909
512 MB PC3200	719	0.0909
1 GB PC3200	3590	1.0909
256 MB Notebook DDR PC3200 PQI	79	0.0909
512 MB Notebook DDR-II PC4200 <533MHz>	79	0.0909
1 GB Notebook DDR-II PC4200 <533MHz>	79	0.0909
MMC Multimedia kártya 128 MB	269	0.0909
MMC Multimedia kártya 256 MB	499	0.0909
MMC Multimedia kártya 512 MB	799	0.0909
MMC Multimedia kártya 1 GB	1390	0.0909
Smart Média 128 MB	369	0.0909
Compact Flash 128 MB	249	0.0909
Compact Flash 256 MB	439	0.0909
Compact Flash 512 MB	740	0.0909
Compact Flash 1 GB	1360	0.0909
Compact Flash 2 GB	2790	0.0909
Compact Flash 2 GB Ultra II	3490	0.0909
SD Secure Digital 128MB	269	0.0909
SD Secure Digital 256MB	549	0.0909
SD Secure Digital 512MB	899	0.0909
SD Secure Digital 1GB	1490	0.0909
XD-Picture 128MB	499	0.0909
XD-Picture 256MB	849	0.0909
XD-Picture 512MB	1250	0.0909
160 GB Freecom FHD-3 7200rpm USB2+FWire	4390	0.0909
250 GB Freecom FHD-3 7200rpm USB2	4700	0.0909
160 GB Western Digital 7200rpm 8MB 3év!	1720	0.0909
200 GB Western Digital 7200rpm 8MB 3év!	2090	0.0909
250 GB Western Digital 7200rpm 8MB 3év! ÚJ!	2980	0.0909
80 GB Samsung 7200rpm Serial ATA 3év!!!	1250	0.0909

10. ábra Számított mezők használata

### 3.3.5 Rekordok kiválasztása

Miután láttuk, hogyan szabályozhatjuk az eredményhalmazban megjelenő mezőket, nézzük meg, milyen lehetőségeink vannak a rekordok kiválogatására.

#### WHERE záradék

Az eredményhalmazban megjelenő rekordok, mezőértékek és darabszám szerint is szabályozhatók. Előbbi esetben az mondjuk meg, melyik rekordokra vagyunk kíváncsiak, utóbbiban azt, hogy hány rekord jelenjen meg.

Ha rekordokat mezőértékek alapján akarunk válogatni, **FROM** záradék után szerepeltetni kell a **WHERE** kulcsszót, amelyet egy logikai kifejezésnek (úgynevezett feltételnek) kell követnie.

#### WHERE logikai\_kifejezés

A logikai kifejezés általában egy **mezőhivatkozásból**, egy **összehasonlító operátorból** illetve további kifejezésből áll. Minden esetben **egy állítást** fogalmaz meg. Az eredményhalmazban csak azok a rekordok jelennek meg, amelyekre a kifejezés állítása igaz értékű.

Az alábbi lekérdezés csak azokat a vevőket sorolja fel, akik Egerben laknak.

```
SELECT * FROM vevo WHERE vVaros='Eger' ;
```

Összehasonlító operátorként felhasználhatjuk a matematikából ismert relációs jeleket, illetve néhány speciális operátort.

Operátor	Jelentés	Példa	Magyarázat
>	Nagyobb mint	tEladAr>50000	
<	Kisebb mint	tBeszerAr<50000	
=	Egyenlő	rSzalit=rSzamla	
<=	Kisebb, vagy egyenlő	tBeszerAr<=50000	
>=	Nagyobb, vagy egyenlő	tBeszerAr>=50000	
<>, vagy !=	Nem egyenlő	rSzalit!=rSzamla	A szállítási cím nem egyezik meg a számlacímmel.
mező Like szöveg	Hasonló	vNev Like „B%” rDatum Like „2010%”	A LIKE operátor segítségével a mező egy szöveggel hasonlítható össze. A szövegben helyettesítő karakterként használhatjuk a '%', és a '_' karaktereket. A '%', tetszőleges számú, bármilyen karaktert, a „_”, pontosan egy, tetszőleges karaktert helyettesíthet.
mező Between ... and ...	Között	tEladAr Between 100 and 1000	Egy intervallum két megadott végpontja között lévő értékek válogathatók ki. Itt például azok a rekordok, ahol dolgozó keresete 100 és 1000 között van. A két végpont része az intervallumnak.

Operátor	Jelentés	Példa	Magyarázat
mező <b>IN</b> (halmaz)	Eleme	<b>idTermek (4;6;12)</b>	<b>IN</b> Az <b>IN</b> jobb oldalán kerek zárójelek között, felsorolással megadott halmaz szerepel. Az elemeket vesszővel választjuk el. Az <b>IN</b> operátorral azt vizsgálhatjuk meg, hogy a mezőértéke eleme-e a felsorolással megadott halmaznak. A példában akkor kapunk vissza rekordokat, ha azok kiválasztott mezőjének értéke 4, 6, vagy 12.
mező <b>IS NULL</b>	Üres	<b>rCsomag IS NULL</b>	Az <b>IS</b> speciális operátor. Jobb oldalán csak az üres mezőt jelző „NULL” érték állhat. A <b>NULL</b> a „semmit”, jelzi. A <b>mező IS NULL</b> logikai kifejezéssel olyan rekordokat válogathatunk ki, amelyekben a kiválasztott mező üres.

Az eddig látott egyszerű logikai kifejezésekből az AND, illetve OR logikai operátorokkal összetett logikai kifejezéseket alakíthatunk ki, amelyek szintén állítások, tehát igaz, vagy hamis értéket vehetnek fel az egyes rekordokban.

Ha L1 és L2 egyszerű logikai kifejezések, az alábbi igazságtáblákból kiolvasható hogy az AND illetve az OR operátorok esetében mikor igaz az L1-és L2-ből alkotott összetett logikai kifejezés értéke.

		L2	
		AND	
L1	I	I	H
	H	H	H

		L2	
		OR	
L1.	I	I	I
	H	H	H

Az alábbi lekérdezés WHERE záradéka összetett logikai kifejezést tartalmaz. Az eredményhalmazban csak azok a **vevo** táblából, amelyekre mindkét egyszerű logikai kifejezés igaz, tehát a **vVaros** mező értéke '**Eger**' ÉS a **vTelefon** mező '**(20)**'-szal kezdődik.

### LIMIT záradék

A LIMIT kulcsszóval nem adhatunk meg feltételt, ellenben szabályozhatjuk, az eredményhalmazban megjelenő rekordok számát. A záradéknak az SQL mondat végén kell elhelyezkednie, használata pedig az alábbi formában lehetséges:

```
LIMIT {[eltolás,] sorok | sorok OFFSET eltolás}
```

A *sorok* értékkel adjuk meg, hogy hány sort szeretnénk megjeleníteni az eredményhalmazban, az *offset*-tel pedig azt szabályozzuk, hogy a forráshalmaz elején hány rekord maradjon ki a listázásból.

Az alábbi lekérdezés az első 20 rekordot követő 10 rekordot jelenít meg a **vevo** táblából.

```
SELECT * FROM vevo LIMIT 10 OFFSET 20
```

A szintaktikai leírásból látható, hogy az eltolás elhagyható. Ha nem használjuk, a listázás a forráshalmaz elején kezdődik.

### 3.3.6 Rekordok sorba rendezése

Az SQL nemcsak a rekordok kiválogatására, hanem az eredményhalmaz rekordjainak mező, vagy számított értékek szerinti, akár többszintű rendezésére is alkalmas. A rendezés szintjeit adó tételeket egymástól vesszővel elválasztva, az ORDER BY záradék után soroljuk fel. Minden tétel lehet mezőhivatkozás, vagy kifejezés. A rendezési szinteket a felsorolás sorrendje szabályozza.

```
ORDER BY rendezési_tétel [,rendezési_tétel...]
```

Az alábbi lekérdezés előbb lakóhely szerint, majd azon belül névsorba rendezi a **vevo** tábla rekordjait.

```
SELECT * FROM vevo ORDER BY vVaros, vNev;
```

Az ORDER BY záradékban a rendezésre használt tétel nevét követő **ASC|DESC** kulcsszavakkal szabályozható a rendezés iránya. Az **ASC** az alapértelmezett, ezért ha nem adunk meg rendezési irányt, akkor a DBMS növekvő sorrendet állít föl (szövegek esetén betűrendet). Ha valamelyik rendezési tétel alapján csökkenő sorrendet szeretnénk, szerepeltessük a tétel után a **DESC** kulcsszót.



Az ORDER BY és a LIMIT záradékok együttes használatával úgynevezett csúcsértékeket választhatunk ki. Ilyenkor arra keresünk választ, hogy melyek azok a rekordok, amelyek valamilyen szempontból a többi elé kerülnek.

Például: melyik a termék tábla 10 legdrágább terméke? A megoldáshoz eladási ár szerint csökkenő sorrendben listázzuk a termékeket, és csak az első 10 rekordot jelenítjük meg:

```
SELECT tMegnevezes, tEladAr FROM termék
ORDER BY tEladAr DESC
LIMIT 10;
```

### 3.3.7 Különböző rekordok kiválogatása

Bár a SELECT formai leírásában szerepel, eddig nem tettünk említést az ALL, a DISTINCT és a DISTINCTROW kulcsszavak jelentéséről. Az ALL kulcsszó az alapértelmezett, ha egyiket sem használjuk az olyan, mintha az ALL-t alkalmaztuk volna. A DISTINCT hatására az eredményhalmazból csak az egymástól különböző rekordok jelennek meg.

A következő lekérdezés minden vevő lakóhelyét (város) kilistázza.

```
SELECT vVaros FROM vevo;
```

A DISTINCT kulcsszóval kiegészített változat csak az egymástól különböző városneveket jeleníti meg.

```
SELECT DISTINCT vVaros FROM vevo;
```

A MySQL-ben a DISTINCTROW ugyanazt a hatást fejti ki, mint a DISTINCT. A két kulcsszó egymás szinonimája.

## 3.4 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 3.4.1 Összefoglalás

Tananyagunk 3. leckéjében a választó lekérdezések készítésére alkalmas SELECT parancs használatát tekintettük át. A SELECT parancs alapvetően egy megadott rekordforrás rekordjaiból (forráshalmaz) rekordok, és mezők kiválogatására, az eredményhalmaz előállítására alkalmas. A SELECT által létrehozott eredményhalmaz nem tárolódik ismét az adatbázisban, csupán a kliens felületén jelenik meg. A DBMS minden egyes alkalommal újra előállítja az eredményhalmazt, még akkor is, ha a kapott lekérdezés teljesen megegyezik egy koráb-

ban már elvégzett SELECT-tel. Így biztosítható, hogy a lekérdezések eredményhalmaza mindig az adatbázis aktuális állapotát tükrözze.

A SELECT használatakor, a parancsot követő mezőhivatkozások vesszővel tagolt felsorolásával adjuk meg az eredményhalmazban megjelenő mezőket, a FROM záradékot követő táblahivatkozással pedig a rekordforrást azonosítjuk.

A SELECT mezőlistájában elhelyezett kifejezésekkel számított mezőket készíthetünk. Ilyenkor szinte mindig álnevet rendelünk az eredményhalmaz megfelelő oszlopához.

A választó lekérdezések eredményhalmazában megjelenő rekordokat szám, és tartalom szerint is szabályozhatjuk. A rekordok számára a LIMIT, tartalmára a WHERE záradékkal adhatunk meg korlátozást.

A WHERE záradékot mindig egy egyszerű, vagy összetett logikai kifejezés követi, amelynek tényezői mezőhivatkozások is lehetnek. Az eredményhalmazban csak azok a rekordok jelennek meg, amelyekre a WHERE után álló logikai kifejezés igaz értékű.

Az eredményhalmaz rekordjainak rendezését az ORDER BY záradékkal írhatjuk elő. A záradékot követő, mezőhivatkozásokból, vagy kifejezésekből álló tételek, a rendezés szintjeit szabályozzák. Alapértelmezés szerint növekvő, a DESC kulcsszó használata esetén csökkenő rendezés valósul meg.

### 3.4.2 Önellenző kérdések

1. Mit nevezünk rekordforrásnak, forráshalmaznak, és mit eredményhalmaznak?
  - A rekordforrás az a tábla (vagy táblakapcsolat), amelynek rekordjaiból egy lekérdezés válogat. A forráshalmaz a rekordforrás rekordjainak halmaza. Az eredményhalmaz a forráshalmazból kiválogatott rekordok és mezők mátrixa.
2. Hogyan adhatunk álnevet az eredményhalmazban megjelenő mezőknek?
  - A lekérdezés mezőlistájában a mezőhivatkozás után meg kell adni az álnevet.
3. Hogyan hozhat létre számított mezőket egy lekérdezésben?
  - A lekérdezés mezőlistájában szereplő kifejezések eredménye megjelenik az eredményhalmazban. Ha egy ilyen kifejezés mezőhivatkozást tartalmaz, akkor az

eredmény minden rekordban az adott mező értékétől függ. A számított mezőket leíró kifejezéshez általában álnevet rendelünk.

4. Hogyan szabályozhatjuk, hogy melyik rekordok jelenjenek meg az eredményhalmazban.
  - A **WHERE** záradék után megadott logikai kifejezéssel.
5. Hogyan jeleníthet meg csúcsértéket?
  - Az **ORDER BY** és a **LIMIT** záradékok együttes alkalmazásával.



## 4. LECKE: KAPCSOLÓDÓ TÁBLÁK REKORDJAINAK KEZELÉSE

### 4.1 CÉLKITÚZÉSEK ÉS KOMPETENCIÁK

Az előző leckében sok mindent megtudtunk a SELECT parancs használatával kapcsolatban. Láttuk, hogyan lehet mezőket kiválasztani a FROM záradék táblahivatkozásával megjelölt rekordforrásból. Megismertük a rekordok kiválogatásának két módszerét, a WHERE záradékkal megadható logikai kifejezések, és a LIMIT záradékkal beállítható darabszám korlátozás használatát. Szó esett a DISTINCT kulcsszóról, melynek segítségével kihagyhattuk az eredményhalmaz ismétlődő rekordjait. Bár az eddig tanultak is gazdag lehetőségeket kínálnak a parancs használatában, a SELECT alkalmazási köre igazából akkor teljesedik ki, amikor a lekérdezés nemcsak egy tábla, hanem egymással összekapcsolt táblák, kapcsolódó rekordjaiból alkotott forráshalmazra vonatkozik.

Mindezidáig csak egy-egy tábla rekordjait kérdeztük le. Tudjuk, hogy a relációs adatbázisok kialakítása közben dekompozícióval, a táblák szétdarabolásával csökkentjük a redundanciát. Ennek általában az az ára, hogy az egyes egyedek összes tulajdonságát csak több tábla áttekintésével tudjuk megvizsgálni. Minta adatbázisunkban külön táblákban helyezkednek el a termékek, és a termékkategóriák. Ha meg akarjuk tudni, hogy egy termék melyik kategóriához tartozik, két táblát is meg kell vizsgálnunk. Az ilyen problémákat oldják meg a több táblás rekordforrást feldolgozó lekérdezések.

Most következő leckénkben azt fogjuk megtanulni, hogyan lehet táblák kapcsolatát rekordforrásnak tekinteni, az ilyen rekordforrás kapcsolódó rekordjaiból forráshalmazt alkotni, majd abból rekordokat kiválogatni.

A leckében az alábbi kérdésekre mindenképpen keressen válaszokat:

- Mit jelent a kapcsolódó tábla, a kapcsolódó mező, és kapcsolódó rekord fogalma?
- Hogyan lehet kettő vagy több kapcsolódó táblából rekordforrást, illetve forráshalmazt kialakítani?
- Milyen technikái vannak a kapcsolatok leírásának?
- Mit jelent, és hogyan alakítható ki a táblák szoros és laza illesztésű kapcsolata?

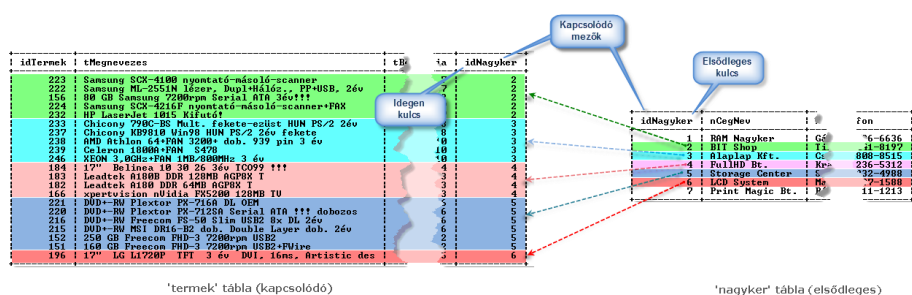
Ha úgy érzi, hogy a következő parancsok szintaktikája túlságosan bonyolult, vagy nehéz azt fejben tartani, a megoldás módja helyett koncentráljon a

céla! Arra gondoljon, mit szeretne elérni. Ha ezt meg tudja fogalmazni, sokkal könnyebb lesz megérteni és megjegyezni az SQL által kínált megoldást.

## 4.2 TÁBLÁK KÖZÖTTI KAPCSOLATOK

Mint tudjuk, a relációs adatmodellben idegen kulcsokkal tároljuk a táblák rekordjai közötti kapcsolatokat. Az idegen kulcs egy tábla olyan mezője, amelynek értékei egy másik (idegen) tábla rekordjait azonosítják. Két olyan táblát, amelyek rekordjai között kapcsolat van, **kapcsolódó táblának** hívunk. A két tábla közötti kapcsolatot az egyik tábla **elsődleges kulcsa** és a másik táblában tárolt **idegen kulcs** segítségével írjuk le. Ezeket a mezőket nevezzük a két tábla **kapcsolódó mezőinek**. A kapcsolatban **elsődleges** táblának nevezzük azt a táblát, amelyiknek elsődleges kulcsa, **kapcsolódó** táblának azt, amelyiknek idegen kulcsa vesz részt a kapcsolatban. A táblák **kapcsolódó rekordjai** azok, amelyekben a kapcsolódó mezők értékei azonosak.

A gyakorláshoz használt **webbolt** adatbázisban 1:N kapcsolat van a **nagyker** és a **termek** táblák között.



11. ábra 'termek' és 'nagyker' táblák kapcsolata

Az ábra ezt az 1:N kapcsolatot ábrázolja. Jól látszik, hogy egy nagykereskedéshez több termék, még egy termékhez csak egy nagykereskedés tartozhat. A **termek** tábla valamennyi rekordja kapcsolódik egy nagykereskedéshez, de **nagyker** táblában van két rekord, amelyekhez egyetlen termék sem kapcsolódik. A két táblának kapcsolódó rekordjai azok, ahol a **nagyker** tábla **idNagyker** mezője, és a **termek** tábla **idNagyker** mezője azonos értéket tartalmaz. Például a Bit Shop nevű cég rekordja kapcsolódik a Samsung SCX-4100 típusú nyomtató rekordjához.

### 4.3 TÖBB TÁBLÁBÓL ÁLLÓ REKORDFORRÁSOK

Tegyük fel, hogy az a feladatunk, hogy megmutassuk, melyik termék melyik nagykereskedésből származik. Ehhez olyan lekérdezést célszerű készítenünk, amely forráshalmazának minden rekordja egy termék és a hozzá kapcsolódó nagykereskedés rekordjából tevődik össze. Az ilyen forráshalmazból már a megismert módon választhatjuk ki a termék és kategória neveket tartalmazó mezőket.

idTermek	nMegnevezes	idNagyker	idNagyker	nCegNev	nKapcsolat	nTelefon
156	80 GB Samsung 7200rpm Serial ATA 360t!!	2	2	BIT Shop	Tibold ábrában	(60) 441-8197
222	Samsung ML-2551M lézer, Dupl+Hálóz., PP+USB, 26v	2	2	BIT Shop	Tibold ábrában	(60) 441-8197
223	Samsung SCX-4100 nyomatógéző-másoló-scanner	2	2	BIT Shop	Tibold ábrában	(60) 441-8197
224	Samsung SCX-4216F nyomatógéző-másoló-scanner+PAX	2	2	BIT Shop	Tibold ábrában	(60) 441-8197
232	HP LaserJet 1015 Kifutó!	2	2	BIT Shop	Tibold ábrában	(60) 441-8197
233	Chicony 790C-BS Mult. fekete-ezüst HUN PS/2 26v	3	3	Alaplap Kft.	Csontos Éva	(30) 888-8515
237	Chicony KB810 Win98 HUN PS/2 26v fekete	3	3	Alaplap Kft.	Csontos Éva	(30) 888-8515
238	AMD Athlon 64+FXN 3200+ dob. 739 pin 3 év	3	3	Alaplap Kft.	Csontos Éva	(30) 888-8515
239	Celeron 1800+FXN 5478	3	3	Alaplap Kft.	Csontos Éva	(30) 888-8515
246	NEON 3,0GHz+FXN 1MB+880MHz 3 év	3	3	Alaplap Kft.	Csontos Éva	(30) 888-8515
166	Expertvision nVidia FXS-200 12000 TU	4	4	PullHD Bt.	Krélik Emma	(70) 236-5312
182	Leadtek 8100 DDR 64MB AGP8X T	4	4	PullHD Bt.	Krélik Emma	(70) 236-5312
183	Leadtek 8100B DDR 128MB AGP8X T	4	4	PullHD Bt.	Krélik Emma	(70) 236-5312
184	17" Belinea 19 30 26 36v IC099 !!!	4	4	PullHD Bt.	Krélik Emma	(70) 236-5312
151	160 GB Freecon FHD-3 7200rpm USB2+FWire	5	5	Storage Center	Seres Dalna	(20) 832-4988
152	80 GB Freecon FHD-3 7200rpm USB2	5	5	Storage Center	Seres Dalna	(20) 832-4988
215	DVD+RW MS1 DR16-B2 dob. Double Layer dob. 26v	5	5	Storage Center	Seres Dalna	(20) 832-4988
216	DVD+RW Freecon FS-50 Slim USB2 8x DL 26v	5	5	Storage Center	Seres Dalna	(20) 832-4988
220	DVD+RW Flexstor PX-71250 Serial ATA 1t dobozos	5	5	Storage Center	Seres Dalna	(20) 832-4988
221	DVD+RW Flexstor PX-7160 DL OEM	5	5	Storage Center	Seres Dalna	(20) 832-4988
196	17" LG L1720P TFT 3 év DVI, 16ms, Artistic de	6	6	LCD System	Marton Ábel	(30) 337-1588

12. ábra Két tábla kapcsolódó rekordjaiból alkotott forráshalmaz

A kérdés ezek után csak az, hogyan tudjuk rávenni a DBMS-t arra, hogy előállítsa a két tábla kapcsolódó rekordjaiból álló forráshalmazt. Mivel az SQL-ben nem kell megadnunk egy feladat megoldásnak módját, csupán az eredményt kell pontosítanunk, a megoldás viszonylag egyszerű.

A SELECT utasítás FROM záradékában úgy kell megadni a rekordforrást, hogy jelezzük, **mely táblák rekordjainak kapcsolatára** vagyunk kíváncsiak, és **melyik mezők biztosítják a rekordok kapcsolatát (kapcsolódó mezők).**

Nos, bár egyszerű megoldást ígértünk, most rövid időre mégis fölládozzuk ezt az egyszerűséget a didaktikus tananyag fölépítés oltárán. Ha az olvasó a következő sorok elolvasása után kissé tanácstalannak érzi magát, kérjük, nyugodjon meg, percekben belül minden letisztul.

A rekordforrásban tehát táblák kapcsolatát kell megadnunk. Ez a kapcsolat lehet szoros (INNER JOIN), illetve laza (OUTER JOIN) illesztésű.

**Szoros illesztés esetén mindkét kapcsolódó táblából csak a kapcsolódó rekordok kerülnek a forráshalmazba (ilyen illesztést láttunk a fenti ábrán).**

**Laza illesztés esetén az egyik táblából az összes, a másik táblából csak a kapcsolódó rekordok kerülnek a forráshalmazba.**

Az esetek többségében szoros illesztést alkalmazunk, ezért a kapcsolatleírás általános formájának bemutatása után ezzel fogunk foglalkozni.

```
SELECT oszlop_meghatározások
FROM
    tábla_1
    INNER JOIN | LEFT OUTER JOIN | RIGHT OUTER JOIN
    tábla_2
ON tábla_1.kapcsolódó_mező = tábla_2.kapcsolódó_mező
```

Két tábla kapcsolatából álló rekordforrást (tábla\_1 és tábla\_2) a fenti formában adhatunk meg a MySQL-ben:

### Táblák kapcsolata:

A **FROM**-ot követően megadjuk a két tábla hivatkozását (nevét) és a közöttük elhelyezett **INNER JOIN**, illetve a kétféle **OUTER JOIN** kulcsszavakkal jelezzük az alkalmazni kívánt illesztési szabályt.

### Kapcsolódó mezők:

Ezek után, az **ON** kulcsszót követően adjuk meg a két tábla kapcsolódó mezőinek hivatkozásait (*tábla\_1.kapcsolódó\_mező*, *tábla\_2.kapcsolódó\_mező*), amelyek közé egyenlőség jelet téve jelezzük, a rekordkapcsolatok feltételét, azaz hogy a két mezőnek azonos értéket kell tartalmaznia a rekordok összekapcsolásához.

Vegyük észre, hogy az **ON** kulcsszót követő két mezőhivatkozás némileg eltér az eddig használttól. A mezőkre eddig egyszerűen a névükkel hivatkoztunk, most pedig a mezőnév előtt feltüntetjük a mezőt tartalmazó tábla nevét is (**tábla\_2.kapcsolódó\_mező**). A tábla és mezőnév közé pontot teszünk. Az ilyen mezőhivatkozást **minősített hivatkozásnak** nevezzük, és olyankor használjuk, amikor ugyanaz a mezőnév a lekérdezés rekordforrásnak több táblájában is szerepel. A minősített hivatkozással **megelőzhető** az úgynevezett **többértelmű hivatkozás**, amikor is a DBMS nem tudja eldönteni, melyik tábla mezőjére gondoltunk.

Azokban a lekérdezésekben, amelyek több tábla kapcsolatán alapulnak célszerű mindig minősített hivatkozással hivatkozni a mezőkre.

A DBMS úgy állítja elő az eredményhalmazt, hogy a **FROM**-ot követően megadott rekordforrás alapján automatikusan elkészíti a forráshalmazt, majd a lekérdezés többi részének megfelelően (oszlop meghatározások, **WHERE**,



**LIMIT)** kiválogatja a kért mezőket és rekordokat. Ezek után lássuk a példákat, amelyek remélhetően mindent világossá tesznek!

### 4.3.1 Szoros illesztés

Szoros illesztésű kapcsolatot használunk akkor, amikor arra vagyunk kíváncsiak, hogy melyek két tábla kapcsolódó rekordjai. Ilyenkor a **FROM** záradékban megadott táblák hivatkozásai közé az **INNER JOIN** kulcsszavakat illesztjük. A forráshalmazba mindkét táblából csak a kapcsolódó rekordok helyezi el a DBMS.

Az alábbi lekérdezés megjeleníti a **termek** és **nagyker** táblák kapcsolódó rekordjait. Figyeljük meg, hogy a rekordforrás egyik táblájából sem kerülnek a forráshalmazba azok a rekordok, amelyek nem kapcsolódnak a másik táblához!

```
SELECT *
FROM termek INNER JOIN nagyker
ON termek.idNagyker=nagyker.idNagyker
```

idTermek	tMegnevezes	tBesz	idNagyker	idNagyker	nCegNev	n	telefon
156	80 GB Samsung 7200rpm Serial ATA 360t!!!	11	2	2	BIT Shop	1	441-8197
222	Samsung ML-2551N lézér, dupl. működés, PP+USB, 2év	123	2	2	BIT Shop	1	441-8197
223	Samsung SCX-4100 nyomtató-szkennelő-scanner	55	2	2	BIT Shop	1	441-8197
224	Samsung SCX-4216FP nyomtató-szkennelő-scanner+PAX	79	2	2	BIT Shop	1	441-8197
232	HP LaserJet 1015 Kifutó!	43	2	2	BIT Shop	1	441-8197
233	Chicony 7902-BS Mult. felh.	4	3	3	888-8515	1	888-8515
237	Chicony KDPB10 Min9R HUN	termek táblából származó mezők	3	3	888-8515	1	888-8515
238	AMD Athlon 64+FXN 3200+ 4	18	3	3	888-8515	1	888-8515
239	Celeron 1800+FXN 8478	18	3	3	888-8515	1	888-8515
246	XEON 3,0GHz+FXN 1MB/800MHz	73	3	3	888-8515	1	888-8515
166	epson/stylino nVidia FX5200 128MB TU	5	4	4	Palldi Bt.	1	236-5312
182	Leadtek 0180 DDR 64MB RGPX T	5	4	4	Palldi Bt.	1	236-5312
183	Leadtek 0180B DDR 128MB RGPX T	5	4	4	Palldi Bt.	1	236-5312
184	17" Belinea 16 3B 26 36v IC099 !!!	20	4	4	Palldi Bt.	1	236-5312
151	160 GB Freecom PNB-3 7200rpm USB2+Vire	47	5	5	Storage Center	1	832-4988
152	250 GB Freecom PNB-3 7200rpm USB2	47	5	5	Storage Center	1	832-4988
215	DUD+RW P51 DRL6-80 dob. Double Layer dob. 2év	14	5	5	Storage Center	1	832-4988
216	DUD+RW Freecom FS-50 Slim USB2 8x DL 2év	5	5	5	Storage Center	1	832-4988
220	DUD+RW Flextor FX-7123R Serial ATA ??? dobozos	19	5	5	Storage Center	1	832-4988
221	DUD+RW Flextor FX-7168 DL OEM	21	5	5	Storage Center	1	832-4988
196	17" LG L1720P TFT 3 év DOL. 16ms. Artistic des	59	6	6	LCD System	1	337-1508

13. ábra Szoros illesztéssel létrehozott lekérdezés

A fenti példában nem adtunk meg **WHERE** záradékot és az oszlop meghatározások helyén is a \* hivatkozást helyeztük el. Így a forráshalmaz minden mezője és rekordja, azaz maga a forráshalmaz jelent meg. Innen azonban már csak egy lépés, hogy a forráshalmazból csak szükséges mezők és rekordok kerüljenek az eredményhalmazba.



Az alábbi lekérdezés már csak a termék megnevezését valamint a nagykereskedő nevét mutatja meg, és csak azokat a rekordokat válogatja ki, ahol a 'BIT Shop' a nagykereskedő.



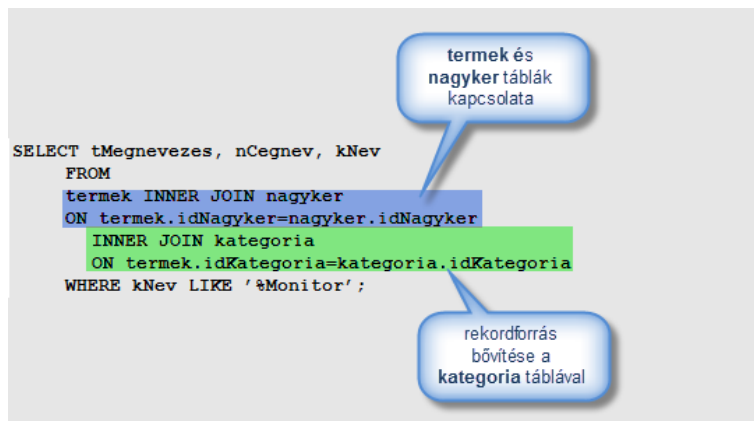
Röviden, megjeleníti a 'BIT Shop'-tól vásárolt termékeket.

```
SELECT tMegnevezes, nCegnev
FROM
    termék
    INNER JOIN
    nagyker
    ON termék.idNagyker=nagyker.idNagyker
WHERE nCegnev='BIT Shop' ;
```

A fenti példának megfelelően bármelyik két tábla kapcsolatát meg tudjuk fogalmazni egy lekérdezésben. Igen ám, de vajon mi a teendő, ha nem kettő, hanem több tábla kapcsolata alkotja a lekérdezés rekordforrását? Ha például látni szeretnénk, hogy kitől vásároltunk monitorokat, a rekordforrást a **termék**, **kategória** és **nagyker** táblák kapcsolataként kell megadni.

Ha több tábla kapcsolatát kell leírunk, akkor az első táblapár leírt kapcsolatát önálló táblahivatkozásnak tekintjük, és ehhez kapcsoljuk a következő táblát. Amennyiben szükséges, a fentieknek megfelelően tovább bővíthetjük a rekordforrást.

```
SELECT oszlop_meghatározások
FROM
    tábla_1-tábla_2_kapcsolata
    INNER JOIN tábla_3
    ON
    tábla_2.kapcsolódó_mező=tábla_3.kapcsolódó_mező;
```



14. ábra Három táblából álló rekordforrás

### Implicit JOIN

A szoros illesztés eddig látott megvalósítását **explicit** (pontosan meghatározott) illesztésnek nevezzük. Az SQL lehetőséget ad a táblák sokak által kedvelt, **implicit** (nem kifejtett) kapcsolására is. Implicit kapcsoláskor a FROM záradékot követően egyszerűen, vesszővel elválasztva felsoroljuk a kapcsolódó táblákat, majd a WHERE záradékban írjuk le a kapcsolódó mezők egyenlőségét.

```
SELECT tMegnevezes, nCegnev
FROM termék, nagyker
WHERE termék.idNagyker=nagyker.idNagyker;
```

Ha implicit kapcsoláskor több táblát kell összekapcsolnunk, akkor a WHERE záradékban összetett logikai kifejezést kell használni.

```
SELECT tMegnevezes, nCegnev, kNev
FROM termék, nagyker, kategoria
WHERE
    termék.idNagyker=nagyker.idNagyker
    AND
    termék.idKategoria=kategoria.idKategoria;
```

### Táblák álnevei

A SELECT parancs tárgyalásakor már említettük, hogy nem csak a mező, hanem a táblahivatkozásokhoz is rendelhetünk álneveket. Ennek éppen itt, a több táblából álló rekordforrások leírásánál lehet jelentősége.

Mint tudjuk, azokban a lekérdezésekben, amelyek rekordforrása több tábla kapcsolatából áll célszerű minősített hivatkozással hivatkozni a mezőkre (*tábla-név.mezőnév*). Ez meglehetősen sok gépeléssel, ennek megfelelően nagyobb arányú hibalehetőséggel jár. Ha a táblákhoz rövidített álnevet rendelünk, akkor a minősített hivatkozásokban használhatjuk ezeket, sőt ezeket kell használnunk a táblanevek helyén.

Az álnevet a **FROM** záradékban a tábla nevét követően adjuk meg.

Az alábbi példában a **termék** tábla a 't', a **nagyker** tábla az 'n' álnevet kapja. Figyeljük meg, hogy álnévvel ellátott táblákra a lekérdezés minden pontján (oszlop meghatározásokban, kapcsolat leírásban, feltételekben) az álnév használatával hivatkozhatunk.

```
SELECT t.tMegnevezes, k.nCegnev
FROM termék t INNER JOIN nagyker k
ON t.idNagyker=k.idNagyker
WHERE k.nCegNev = 'BIT Shop';
```

## 4.4 LAZA ILLESZTÉSŰ KAPCSOLATOK

A szoros illesztés alkalmas arra, hogy lekérdezzük két (vagy több) tábla kapcsolódó rekordjait. a



*Ezzel az illesztési szabállyal választ kapjuk az olyan típusú kérdésekre, mint:*



*Melyek az 'A' és tábla 'B' tábla azon rekordjai, amelyek kölcsönösen kapcsolódnak egymáshoz?*



*Például melyik nagykereskedés melyik termékeket, és melyik terméket melyik nagykereskedés szállítja.*

A leggyakrabban ilyen kérdéseket kell feltennünk az adatbázis-kezelő rendszernek. Szoros illesztéssel azonban nem tudjuk megkérdezni a következőt: melyek az 'A' tábla azon rekordjai, amelyek **nem kapcsolódnak** a 'B' táblához? Például, melyik az a nagykereskedés, amelyiktől egyetlen terméket sem vásároltunk. Nos, ezt a problémát oldja meg a laza illesztés.

Laza illesztés esetén az **egyik tábla minden rekordja** bekerül a forráshalmazba (függetlenül attól, hogy van-e hozzá kapcsolódó rekord a másik táblában). A másik táblából azonban **csak a kapcsolódó rekordok** kerülnek be.



*A nagykereskedések termékek példájában ez azt jelenti, hogy minden nagykereskedés rekordja a forráshalmazba kerül, de csak azokhoz kapcsolódik a termék táblából származó rekord, amelyik valóban szállít valamilyen terméket.*

A laza illesztés lehet, bal és jobb oldali, attól függően, hogy a bal oldali, vagy a jobb oldali táblából akarunk minden rekordot megjeleníteni. A kapcsolat leírását a táblahivatkozások közé írt **LEFT OUTER JOIN**, vagy **RIGHT OUTER JOIN** kulcsszavakkal adhatjuk meg. (Az **OUTER** szó mindkét esetben elhagyható.)

Az alábbi lekérdezésben a jobb oldali, **nagyker** tábla összes rekordja, a bal oldali **termék** táblának csak a kapcsolódó rekordjai kerülnek a forráshal-

mazba. Az eredményhalmazban jól látható, hogy az első és utolsó nagykereskedéshez nem kapcsolódik termék.

```
SELECT *
FROM termek t RIGHT JOIN nagyker n
ON t.idNagyker=n.idNagyker;
```

idTermek	tNevnevezes	tBeszeres	tEladrs	idNagyker	idNagyker	nCegNev	nKapcsolat	nTelefon
NULL	NULL	NULL	NULL	NULL	1	RAM Nagyker	Gli Oszkár	(70) 626-6636
156	80 GB Samsung 7200rpm Serial ATA 360***	12580	13838	2	2	BIT Shop	Tibold ábrában	(60) 441-0197
222	Samsung ML-2551N lézer, Dupl+Háldo., PP+USB, 26w	129900	142890	2	2	BIT Shop	Tibold ábrában	(60) 441-0197
223	Samsung SKC-4100 nyomató-néző-scaner	55900	61490	2	2	BIT Shop	Tibold ábrában	(60) 441-0197
224	Samsung SKC-4216F nyomató-néző-scaner+FXR	79990	87989	2	2	BIT Shop	Tibold ábrában	(60) 441-0197
228	DUD+RU Flexstor PK-7125A Serial ATA *** dabozos	19500	21450	5	5	Storage Center	Seres Balna	(20) 832-4908
221	DUD+RU Flexstor PK-716A DL ODM	21900	24090	5	5	Storage Center	Seres Balna	(20) 832-4908
196	17" LG L172HP TFT 3 ów DVI, 16m, Artistic des	59900	65890	6	6	LCD System	Marton ábel	(30) 337-1508
NULL	NULL	NULL	NULL	NULL	7	Print Magic Bt.	Rács Ilona	(30) 811-1213

15. ábra Jobb oldali laza illesztés a termék és nagyker táblák között

Ha kicsit elgondolkodunk, azonnal fölmerül bennünk a kérdés, hogy vajon milyen értékek szerepelnek majd azokban a mezőkben, amelyek a terméket nem szállító nagykereskedések mellett, a **termek** táblából származnának...

A fenti ábrán, az első és utolsó nagykereskedéshez nem kapcsolódik termék (nyilvántartjuk őket, de még nem vásároltunk tőlük). Jól látszik, hogy ezekben a rekordokban **NULL** értékek szerepelnek a **termek** táblából származó mezőkben.

Ezt tudva, a laza illesztés alkalmassá válik arra, hogy csak azokat a rekordokat válogassuk ki, ahol a nagykereskedéstől még nem vásároltunk, azaz a nagykerhez nem kapcsolódik egyetlen termékünk sem. Ehhez csupán egy **IS NULL** feltételt kell megadnunk valamelyik **termek** mezőre.

```
SELECT *
FROM termek t RIGHT JOIN nagyker n
on t.idNagyker=n.idNagyker
WHERE t.idTermek IS NULL;
```

idTermek	tNevnevezes	tBeszeres	tEladrs	tKeszlet	tSzukseges	idKategoria	idNagyker	idNagyker	nCegNev	nKapcsolat	nTelefon
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	1	RAM Nagyker	Gli Oszkár	(70) 626-6636
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	7	Print Magic Bt.	Rács Ilona	(30) 811-1213

16. ábra NULL értékeket tartalmazó rekordok kiválasztása

```
SELECT n.*
FROM termék t RIGHT JOIN nagyker n
  on t.idNagyker=n.idNagyker
WHERE t.idTermek IS NULL;
```

idNagyker	nCegNev	nKapcsolat	nTelefon
1	RAM Nagyker	Gál Oszkár	<70> 626-6636
?	Print Magic Bt.	Rácz Ilona	<30> 811-1213

17. ábra Csak a szükséges mezők megjelenítése

### NATURAL JOIN

Érdekes lehetőség az úgynevezett **NATURAL JOIN**, ami a szoros, vagy laza illesztés egyik egyszerűsített megvalósítására alkalmas. Alkalmazásakor nem kell leírunk a kapcsolódó mezőket, a DBMS ugyanis megkeresi a kapcsolódó táblák azonos nevű mezőit, és ezek egyenlősége alapján építi fel a kapcsolatot.

```
SELECT tMegnevezes, nCegnev
FROM termék NATURAL JOIN nagyker;
```

A **NATURAL JOIN** egyszerű táblakapcsolást tesz lehetővé, azonban csak akkor működik, ha a kapcsolódó mezők nevei valóban megegyeznek. Ez bizony nem mindig van így, ezért ez a lehetőség korlátozottan használható.

## 4.5 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 4.5.1 Összefoglalás

Mostani leckénkben az SQL talán legizgalmasabb lehetőségét a több táblából álló rekordforrásokon alapuló lekérdezések készítését ismertük meg. Az ilyen lekérdezések elengedhetetlenek a redundancia csökkentése érdekében általában sok táblára darabolt adatbázissal végzett munkában. Mint láttuk, az ilyen lekérdezések készítésekor pontosan meg kell adnunk a rekordforrást alkotó táblákat, a közöttük lévő illesztési szabályt, és meg kell neveznünk a kapcsolódó mezőket. Mindezt megtehetjük a FROM záradék megfelelő használatával.

A záradék után meg kell adnunk a táblák hivatkozásait, amelyeket, az illesztésnek megfelelően **INNER JOIN**, **LEFT OUTER JOIN**, vagy **RIGHT OUTER JOIN** kulcsszavakkal kell elválasztanunk.

Ezután az **ON** kulcsszót követően, minősített hivatkozással kell megneveznünk a kapcsolódó mezőket. A mezőhivatkozások közé egyelőség jelet téve jelezzük a rekordkapcsolatok feltételét. A minősített hivatkozások használatát a rekordforrás tábláihoz rendelt álnevekkel egyszerűsíthetjük.

A leckében megtanultuk, hogy a szoros illesztés megadható explicit, és implicit (táblahivatkozások felsorolása, kapcsolódó mezők a WHERE záradékban) módon is.

Előbbi esetben a

```
FROM tábla1 INNER JOIN tábla2 ON  
tábla1.mező1=tábla2.mező2
```

formátumot, utóbbi esetben a

```
FROM tábla1, tábla2 WHERE tábla1.mező1=tábla2.mező2
```

formátumot használjuk.

Amennyiben a két tábla kapcsolódó mezői azonos nevet kaptak, a **NATURAL JOIN** kulcsszavakkal jelentősen egyszerűsíthetjük a kapcsolat leírását.

Míg a szoros illesztéssel két tábla kapcsolódó rekordjait választhatjuk ki, a laza illesztés arra ad lehetőséget, hogy egy tábla azon rekordjait jelenítsük meg, amelyekhez a másik táblából egyetlen rekord sem kapcsolódik. Laza illesztés esetén **LEFT OUTER JOIN** kulcsszavakkal jelezzük a bal oldali, **RIGHT OUTER JOIN**-nal a jobb oldali laza illesztést.

### 4.5.2 Önellenző kérdések

1. Mit kell világossá tennünk két tábla kapcsolatából álló rekordforrás leírásakor?

- Meg kell adnunk a kapcsolódó táblák és a kapcsolódó mezők nevét, valamint jeleznünk kell a táblák közötti illesztési szabályt.

2. Miért mondhatjuk, hogy az alábbi lekérdezés forrás- és eredményhalmaza azonos?

```
SELECT *
```

```
FROM
```

```
nagyker n INNER JOIN termék t ON n.idNagyker=t.idNagyker
```

- Azért, mert a forráshalmaz összes mezője, és rekordja megjelenik az eredményhalmazban, így a két mátrix azonos tartalmú.

3. Hol tárolja a DBMS a lekérdezések eredményhalmazát?

- Sehol. A DBMS elküldi a kliensnek a lekérdezés eredményhalmazát, a kliens pedig megjeleníti azt a felhasználó előtt. Azonos lekérdezés többszöri végrehajtásakor a DBMS mindig újra létrehozza az eredményhalmazt, így az biztosan az adatbázis aktuális állapotát tükrözi.

4. Mit jelenít meg az alábbi lekérdezés?

```
SELECT vNev,r.idRendeles, rDatum, tMegnevezes, DarabSzam
FROM
vevo v
INNER JOIN rendeles r ON v.idVevo=r.idVevo
INNER JOIN rendelestermek rt ON r.idRendeles= rt.idRendeles
INNER JOIN termék t ON rt.idTermek=t.idTermek
WHERE v.idVevo=4
ORDER BY r.idRendeles
```

- A lekérdezés rekordforrása a vevo-rendeles-rendelesstermek-termek táblák szoros illesztésű kapcsolata. A forráshalmazt az egyes vevők, a hozzájuk kapcsolódó megrendelések, azok tételei és a tételeknek megfelelő termékek kapcsolódó rekordjai alkotják. Az eredményhalmazban csak a vevő neve, a rendelés azonosítója és dátuma, a termék neve és megrendelt darabszáma jelenik meg. Csak azok a rekordok lesznek az eredményhalmazban, ahol a vevő azonosítója 4-es. A rekordok a rendelés azonosítójának megfelelően növekvő sorrendbe kerülnek.

Rövidebben: a 4-es azonosítójú vevő neve, megrendeléseinek azonosítói, dátumai, valamint megrendelt termékek neve, és mennyisége jelenik meg, a rendelés azonosítók szerint rendezett formában.

5. Mi lesz az alábbi lekérdezés eredménye?

```
SELECT v.vNev, v.vTelefon
```



```
FROM  
vevo v LEFT JOIN rendeles r  
ON v.idVevo=r.idVevo  
WHERE r.idVevo IS NULL  
ORDER BY v.vNev
```

- Azoknak a regisztrált vevőknek a nevét és telefonszámát fogjuk látni, akik még nem rendeltek semmit. A lista névsorban jelenik meg.



## 5. LECKE: FÜGGVÉNYEK HASZNÁLATA AZ ADATKEZELÉSBEN

### 5.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

A választó lekérdezésekről szóló leckében megismerkedtünk a számított mezők készítésének technikájával. Akkor egyszerű kifejezésekkel adtuk meg a mező kiszámításának módját. A bonyolultabb számításokat a számított érték előállításához használt kifejezések egyszerű eszközeivel nehezen tudnánk elvégezni. Ilyenkor nyújthatnak hasznos szolgálatot az SQL nyelv beépített számításai, az úgynevezett függvények.

Most következő leckénkben a MySQL függvényeivel foglalkozunk. Látni fogjuk, hogy a függvények különböző típusú adatokat dolgoznak fel, illetve állítanak elő. Ez alapján tipizálva megkülönböztetjük a szöveges, dátum-idő, matematikai és egyéb függvényeket. Leckénkben előbb a legfontosabb, szövegkezelésre alkalmas számításokat, majd a dátumokkal, időpontokkal dolgozó függvényeket, végül a matematikai funkciókat tekintjük át.

A csoportokba sorolt függvények mindegyikénél rövid, a függvény használatának módját leíró specifikációt, illetve néhány példát bemutató leírást közlünk.

A lecke tanulása során ne a felsorolt függvények „bemagolására”, hanem azok áttekintésére, a példák megértésére koncentráljon. A tényleges adatbázis-kezelő feladatokkal találkozva emlékezni fog arra, hogy már olvasott a megoldáshoz esetleg alkalmazható függvényről. A tananyagba visszaolvasva, vagy a MySQL függvényreferenciájába betekintve fel tudja eleveníteni az éppen szükséges függvény formai leírását, specifikációját. A függvények ismeretét, és a használat módját a gyakorlat hozza majd meg. Ennek ellenére a lecke végén természetesen talál majd önellenőrző kérdéseket, amelyek a leggyakoribb függvények használatát firtatják.

### 5.2 A FÜGGVÉNYEKRŐL ÁLTALÁBAN

Az adatbázis-kezelés során gyakran előfordul, hogy nem közvetlenül a tárolt adatokat kell megjelenítenünk, hanem azokkal különböző számításokat, formázásokat, átalakításokat végezve, a műveletek eredményeivel kell tovább dolgoznunk. Az sem ritka, hogy a felhasználó által megadott adatok nem felelnek meg az adatbázis szerkezetének, a mezők típusának. Ezért megfelelő átalakítással meg kell előzni, hogy az adattárolás az SQL-mondatban elhelyezett hibás adat miatt meghiúsuljon.

Az adatok átalakításával, formázásával, az azokkal végezhető számítások lebonyolításával kapcsolatos feladatokat jelentősen megkönnyítik a MySQL függvényei. Ezeket a függvény nevével és szükség esetén az azt követő kerek zárójelek között elhelyezett, vesszővel elválasztott paraméterek átadásával lehet meghívni. Az DBMS elvégzi a számítást és az SQL-mondatban, a hívás helyére illeszti be a függvény eredményét, az úgynevezett visszatérési értéket. Éppen ezért függvényt csak olyan helyen lehet meghívni, ahol a visszatérési értéket fel is használjuk. Függvényhívás elhelyezkedhet számított mező értékét kiszámoló kifejezés tényezőjeként, WHERE záradékot követ logikai kifejezés elemeként, vagy felhasználó változó értékadásában.

Rövidesen látni fogjuk, hogy egy függvény meghívásához nem csak azt kell tudnunk, hogy milyen számítást végez, hanem azt is, hogy mennyi, és milyen paramétert vár.

A függvények bemutatásakor megadjuk azok formai leírását, specifikációját, amely tartalmazza a függvény úgynevezett formális paraméterlistáját. A formális paraméterlista határozza meg a várt paraméterek számát, sorrendjét, és típusát. A függvény hívásakor a formális paraméterlistának megfelelő aktuális paramétereket kell átadnunk. Ezt nevezzük aktuális paraméterlistának.

A függvények paraméterei lehetnek literálok, mezőhivatkozások, vagy más függvények hívásai.

Amikor a paraméter egy forráshalmaz valamely mezőjének hivatkozása, akkor a DBMS a mezőértékét helyettesíti a paraméter helyére. Ha a paraméter egy másik függvény hívása, akkor előbb az értékelődik ki, majd a visszatérési érték a paraméter helyére kerül.

## 5.3 SZÖVEGKEZELŐ FÜGGVÉNYEK

A szövegkezelő függvények szöveges paramétereket dolgoznak fel, vagy visszatérési értékük, eredményük szöveges típusú.

### 5.3.1 Szöveg csonkolása

Szövegek rögzítésekor előfordul, hogy szöveg túl hosszú, vagy olyan szóköz karakterek fogják közre, amelyeket nem szeretnénk tárolni. A tárolt szövegek megjelenítésekor is lehetséges, hogy a szövegnek csak egy részére vagyunk kíváncsiak. A fenti feladatok elvégzésére alkalmasak a szöveg csonkolását lebonyolító függvények.

### LTRIM, RTRIM

Az LTRIM és RTRIM függvények paramétere és visszatérési értéke egyaránt szöveg. A függvények a paraméter bal illetve jobb oldaláról vágják le az esetleges szóközöket és a csonkolt szöveget adják vissza eredményként.

```
LTRIM (szöveg)
RTRIM (szöveg)
```

```
SELECT LTRIM('      Monitor');
+-----+
| Monitor |
+-----+
```

### LEFT, RIGHT, MID

A LEFT, RIGHT, MID függvények a paraméterként kapott szöveg egy részét adják vissza eredményként. A LEFT a szöveg bal, a RIGHT a jobb oldaláról, a MID a paraméterként megadott sorszámtól kezdve ad vissza valahány karaktert.

```
LEFT (szöveg,karakterszám)
RIGHT (szöveg,karakterszám)
MID (szöveg,sorszám,karakterszám)
```

```
SELECT
LEFT('1234567890',4),RIGHT('1234567890',4),MID('1234567890',5,4);
```

```
+-----+-----+-----+
| LEFT('1234567890',4) | RIGHT('1234567890',4) | MID('1234567890',5,4) |
+-----+-----+-----+
| 1234                | 7890                  | 5678                  |
+-----+-----+-----+
1 row in set (0.00 sec)
```

18. ábra LEFT, RIGHT, MID függvények használata

### 5.3.2 Szöveg módosítása

A következő függvények megváltoztatják a paraméterként kapott szöveg tartalmát, és visszaadják az eredményt.

#### LPAD, RPAD

Az LPAD, RPAD függvények a paraméterként kapott szöveg bal, vagy jobb oldalára, egy szintén paraméterként átadott kiegészítő szöveg többszörös előfordulását illesztik. Eredményként a középső paraméterben megadott karakter-hosszúságú szöveget kapunk.

```
LPAD (szöveg,hossz,kiegészítő)
RPAD (szöveg,hossz, kiegészítő)
```

```
SELECT LPAD ('010011',8,'0');
```

```
+-----+
| LPAD('010011',8,'0') |
+-----+
| 00010011             |
+-----+
1 row in set (0.00 sec)
```

19. ábra LPAD

#### CONCAT, CONCAT\_WS

A CONCAT függvény a paraméterként kapott tetszőleges számú szövegből egyetlen karakterláncot készít. A CONCAT\_WS mindezt azzal egészíti ki, hogy az összefűzött szövegrészek közé paraméterként megadható elválasztó karaktert illeszt.

```
CONCAT (szöveg1, szöveg2,...)
CONCAT_WS (elválasztó, szöveg1, szöveg2,...)
```

```
SELECT CONCAT('A','B','C'),
CONCAT_WS(' ','A','B','C');
```

CONCAT('A','B','C')	CONCAT_WS(' ','A','B','C')
ABC	A,B,C

1 row in set (0.00 sec)

20. ábra CONCAT és CONCAT\_WS

## REPLACE

A REPLACE függvény három szöveparamétert vár. Az első szövegben felkutatja a második szöveg előfordulásait, és a harmadik szövegre cseréli azokat.

```
REPLACE(szöveg,mit ,mire)
```

```
SELECT REPLACE('2012/06/06','/',':');
```

REPLACE('2012/06/06','/',':')
2012:06:06

1 row in set (0.00 sec)

21. ábra REPLACE

## 5.3.3 Szöveg keresése

### LOCATE

A LOCATE függvény az első paraméterként megadott szöveget keresi a második paraméterként megadott szövegben és visszaadja az első előfordulás pozícióját.

```
LOCATE (szövegrész,szöveg)
LOCATE (szövegrész,szöveg,kezdet)
```

Ha harmadik paraméterként megadunk egy számot, akkor a keresést az adott karakterpozícióban kezdi.

```
SELECT LOCATE ('@', 'x.y@posta.smtp.hu');
```

```
+-----+
|                                     4 |
+-----+
```

```
SELECT LEFT ('x.y@posta.smtp.hu',
            LOCATE ('@', 'x.y@posta.smtp.hu') - 1);
```

```
+-----+
| x.y
+-----+
```

### 5.3.4 Szöveg formázása

#### LCASE, UCASE

Az LCASE függvény a kisbetűs szöveg nagybetűsre alakítását, a UCASE fordított formázást végez. A két függvény csak a betű karakterekre van hatással.

```
UCASE (szöveg)
```

```
LCASE (szöveg)
```

```
SELECT UCASE ('ac-48153');
```

```
+-----+
| AC-48153
+-----+
```

### 5.3.5 Szöveg hosszának meghatározása

#### LENGTH

A LENGTH függvény a paraméterként kapott szöveg bájtokban mért hosszát adja eredményként. Mivel a hagyományos karakterkódolás esetén a karakter egy bájt a LENGTH eredmény egyben a paraméter karakterekben mért hossza is.

```
LENGTH (szöveg)
```



```
SELECT LENGTH('x.y@posta.smtp.hu');
```

```
+-----+
|                17 |
+-----+
```

```
SELECT
```

```
RIGHT('x.y@posta.smtp.hu',LENGTH('x.y@posta.smtp.hu') -
LOCATE('@','x.y@posta.smtp.hu'));
```

```
+-----+
-----
| posta.smtp.hu
+-----+
-----
```

### CHAR\_LENGTH

UTF8 karakterkódolás esetén előfordulhat, hogy a LENGTH() függvény nem a várt eredményt adja. Ez azért lehetséges, mert az UTF8-ban kódolt szövegek nemzeti karakterei nem egy bájtosak. Ha UTF8 esetén is pontosan szeretnénk megkapni a karakterszámot, a LENGTH() helyett használjuk a CHAR\_LENGTH() függvényt! Ez mindig a paraméterszövegben lévő karakterek számát adja vissza.

## 5.4 DÁTUM- ÉS IDŐKEZELŐ FÜGGVÉNYEK

A dátum- és időkezelő függvények paraméterei, és/vagy visszatérési értékük dátum, illetve idő típusúak. Használatukhoz tudnunk kell, hogy a MySQL az aposztrófok között, 'YYYY-HH-NN', vagy 'YYYY.HH.NN' formában leírt szöveget dátumnak, az 'ÓÓ:PP:MM' formában megadott szöveget pedig időnek tekinti.

A dátum- és időkezelő függvények paraméterei ilyen típusú mezők is lehetnek, sőt, az is gyakori, hogy egy dátumkezelő függvény paramétere egy másik dátumkezelő függvény visszatérési értéke.

Egyes dátum-idő függvények úgynevezett TIMESTAMP formátumban kezelik a dátumokat és időpontokat. A TIMESTAMP 14, 12, 8 vagy 6 karakteren hosszúságú „időbélyező”, amely elválasztó jelek nélküli dátumokat, időpontokat ír le. A TIMESTAMP-ek az alábbi formátumok szerint épülhetnek fel:

14 karakter:           ÉÉÉÉHHNNÓÓPPMM,

12 karakter:           ÉÉHHNNÓÓPPMM

8 karakter:            ÉÉÉÉHHNN

6 karakter:            ÉÉHHNN

### 5.4.1 Munka a rendszeridővel

Elsőként tekintsük át azokat a függvényeket, amelyekkel a számítógépünk által tárolt rendszeridő kezelhető. Ezek a függvények sokat segíthetnek például akkor, ha egy rendelés leadásának, vagy a felhasználó regisztrációjának időpontját, esetleg egy korábbi esemény és a jelenlegi idő között eltelt napok számát akarjuk vizsgálni.

#### NOW, SYSDATE

Ezeknek a függvények nincs szükségük paraméterekre, mindkettő az aktuális időpontot adja eredményül. A visszatérési érték általában 'YYYY-MM-DD HH:MM:SS' formátumú, de ha a függvény matematikai kifejezés része, akkor valós szám, amelynek egész része a TIMESTAMP típusnak megfelelően épül fel.

A NOW() és a SYSDATE() közötti különbség az, hogy még a NOW() a befoguló SQL-mondat végrehajtásának kezdeti, addig a SYSDATE() a meghívásának tényleges időpontját adja eredményül. Az alábbi példa egy speciális függvény a SLEEP() meghívásával demonstrálja a különbséget.

A SLEEP() nem ad vissza eredményt, de visszatérés előtt a paraméterben kapott másodpercig vár.

```
SELECT NOW() , SLEEP(10) , SYSDATE() ;
```

NOW<>	SLEEP<10>	SYSDATE<>
2012-06-13 17:52:07	0	2012-06-13 17:52:17

1 row in set (10.00 sec)

22. ábra NOW(), SLEEP(), SYSDATE()

### 5.4.2 Dátum és idő részeinek feldolgozása

Dátum, vagy időpont kezelésekor előfordulhat, hogy nem az adat egészére, hanem annak egy részére vagyunk csak kíváncsiak.



*Például a leadási dátumból a hónapot kiemelve, havi bontásban szeretnénk látni az egy évben kapott rendeléseket.*

A most következő függvények az ilyen feladatokban segíthetnek.

### YEAR(), MONTH(), DAY(), HOUR(), MINUTE(), SECOND()

A YEAR(), MONTH(), DAY() függvények mindegyike dátumot, VAGY dátum-időt, a HOUR(), MINUTE(), SECOND() függvények pedig dátum-időt, vagy időpontot várnak paraméterként. Eredményük a függvény nevének megfelelő számérték.

**YEAR** (dátum)

**MONTH** (dátum)

**DAY** (dátum)

**HOUR** (dátum-idő)

**MINUTE** (dátum-idő)

**SECOND** (dátum-idő)

```
SELECT YEAR(NOW()) É, MONTH(NOW()) H, DAY(NOW()) N,
       HOUR(NOW()) Ó, MINUTE(NOW()) P, SECOND(NOW()) MP;
```

É	H	N	Ó	P	MP
2012	6	13	17	54	41

1 row in set (0.00 sec)

23. ábra Dátum részletei

### 5.4.3 UNIX\_TIMESTAMP

A MySQL-ben használt TIMESTAMP eltér a Unix operációs rendszerek időbélyeg formátumtól. A Unix TIMESTAMP az 1970-01-01 00:00:00 óta eltelt, másodpercekben megadott idő. Mivel előfordul, hogy a MySQL adatbázisban Unix formátumban kapott dátumot kell eltárolni, szükségünk lehet a Unix típusú időbélyeg kezelésére, illetve a formátumok átalakításra is.

#### UNIX\_TIMESTAMP, FROM\_UNIXTIME

A UNIX\_TIMESTAMP() függvény használható paraméter nélkül, és egy dátumidő típusú paraméter megadásával is. Első esetben az aktuális Unix-időt adja vissza, a másodikban pedig az adott dátumot fejezi ki Unix-idő formájában. A FROM\_UNIXTIME() paramétere Unix-időbélyeg. A függvény MySQL dátum-idő formátumára konvertálja a bemenő adatot.

```
UNIX_TIMESTAMP()  
UNIX_TIMESTAMP(dátum_idő  
)
```

```
FROM_UNIXTIME(UNIX_időbélyeg  
)
```

```
SELECT NOW(), UNIX_TIMESTAMP() U,  
FROM_UNIXTIME(UNIX_TIMESTAMP()) FU;
```

NOW()	U	FU
2012-06-13 17:55:59	1339602959	2012-06-13 17:55:59

1 row in set (0.00 sec)

24. ábra Unix-idő

#### 5.4.4 Műveletek dátumokkal

A dátumokkal való számolást számos függvény támogatja. Ezek lehetőséget teremtenek arra, hogy egy dátumból elvegyünk, vagy ahhoz hozzáadjunk valamekkora időszakot.

##### DATEDIFF, TIMEDIFF

A DATEDIFF() függvény a *dátum1* paraméterből kivonja a *dátum2* paramétert, és napokban adja vissza az eredményt. A TIMEDIFF hasonlóan dolgozik, de az eredményt HH:PP:SS formában adja meg.

```
DATEDIFF(dátum1, dátum2)
```

```
SELECT DATEDIFF(NOW(), '2012.01.01');  
+-----+  
| 157 |  
+-----+
```

##### DATE\_ADD, DATE\_SUB,

A DATE\_ADD() és DATE\_SUB() függvények kicsit kifinomultabbak. Mindkét függvény három paramétert vár. Az első egy dátum, illetve időpont, a második az időegység szorozója, a harmadik az időegység neve. A függvények az első pa-

raméterből kivonják, vagy ahhoz hozzáadják az időegység szorzóval megszorozott értékét. Az eredményt dátum-idő formátumban kapjuk vissza. A használható dátum- és időegységek neveit az alábbi táblázat tartalmazza:

Időegység	Jelentés
MICROSECOND	Milliomod másodperc
SECOND	Másodperc
MINUTE	Perc
HOURL	Óra
DAY	Nap
WEEK	Hét
MONTH	Hónap
QUARTER	Negyedév
YEAR	Év

**DATE\_ADD** (dátumidő, INTERVAL szorzó időegység)

**DATE\_SUB** (dátumidő, INTERVAL szorzó időegység)

Az alábbi példában 6 hónapot adunk a '2012.01.01' dátumhoz.

```
SELECT DATE_ADD('2012.01.01', INTERVAL 6 MONTH);
+-----+
| 2012-07-01 |
+-----+
```

### 5.4.5 Dátumok formátumának szabályozása

A MySQL DATE\_FORMAT() függvénye a dátumok különböző formátumban történő megjelenítését teszi lehetővé. A függvény mindössze két paramétert vár. Az első egy dátum, a második a formátumot leíró, speciális jelölésekből álló szöveg.

**DATE\_FORMAT** (dátum, formátum)

Formátum leíró jelölések:

Szöveg	Leírás
%a	A hét napjának rövidített neve (Sun..Sat)
%b	A hónap rövidített neve (Jan..Dec)
%c	Hónap számmal (0..12)
%D	A nap sorszáma angol képzővel (0th, 1st, 2nd, 3rd, ...)
%d	A hónap napja számmal, két karakteren (00..31)

Szöveg	Leírás
%e	A hónap napja számmal (0..31)
%f	Milliomod másodperc (000000..999999)
%H	Óra (00..23)
%h	Óra (01..12)
%I	Óra (01..12)
%i	Perc, két számjeggyel(00..59)
%j	Az év napja (001..366)
%k	Óra (0..23)
%l	Óra (1..12)
%M	Hónap neve (January..December)
%m	Hónap, két számjeggyel (00..12)
%p	AM vagy PM
%r	12 órás idő, (hh:mm:ss AM or PM)
%S	Másodperc (00..59)
%s	Másodperc (00..59)
%T	24 órás idő (hh:mm:ss)
%U	Hét (00..53), ahol a vasárnap a hét első napja
%u	Hét (00..53), ahol a hétfő a hét első napja
%V	Hét (01..53), ahol a vasárnap a hét első napja; %X-el együtt használható
%v	Hét (01..53), ahol a hétfő a hét első napja; %x-el együtt használható
%W	A hét napjának neve (Sunday..Saturday)
%w	A hét napjának száma (0=Sunday..6=Saturday)
%X	A hét éve, amelyben vasárnap a hét első napja. Az eredmény négy számjegyes szám; a %V-vel együtt használható
%x	A hét éve, amelyben hétfő a hét első napja. Az eredmény négy számjegyes szám; a %v-vel együtt használható
%Y	Év, négy számjeggyel
%y	Év, kettő számjeggyel

```
SELECT DATE_FORMAT(NOW(), '%Y %M %e. %I:%i:%s %p');
+-----+
| 2012 June 6. 03:07:30 PM |
+-----+
```

## 5.5 MATEMATIKAI FÜGGVÉNYEK

Azt hinnénk, hogy adatbázis-kezelés gyakorlatában viszonylag ritkán van szükségünk matematikai függvények SQL-mondatokkal történő végrehajtására. Elég azonban a számértékek kerekítésre gondolnunk és máris beláthatjuk, hogy bizony szükségünk lehet az SQL matematikai műveleteket támogató funkcióira. A lelkünk további részében a legfontosabb műveleteket, a kerekítéseket, és az előjelkezelést támogató függvények használatát mutatjuk be, majd kitérünk a véletlen számok és a globális azonosító generálásának kérdésére.

### 5.5.1 Egyszerűbb matematikai számítások

A MySQL számos függvénnyel támogatja az olyan egyszerűnek tűnő, de SQL-ben azért igen csak nehezen elképzelhető feladatok végrehajtását, mint a maradékos osztás, a gyökvonás, vagy a hatványozás.

#### PI meghatározása

A PI érték meghatározását a paraméter nélküli PI() függvénnyel végezhetjük el.

#### Maradékos osztás

A maradékos osztásra használható MOD() függvény két számparamétert vár. Az első elosztja a másodikkal és a maradékkal tér vissza.

```
MOD (szám1, szám2)
```

#### Hatványozás

A hatványozás a kétparaméteres POW() függvénnyel végezhető el. A függvény az első számparamétert a másodikként megadott számnak megfelelő hatványra emeli.

```
POW (szám1 , szám2)
```

### Gyökvonás

A gyökvonásra használható SQRT függvény egyetlen pozitív számot vár paraméterként, eredménye pedig a szám négyzetgyöke. Negatív paraméterre NULL értéket kapunk.

```
SQRT (szám)
```

## 5.5.2 Számok előjelének kezelése és kerekítés

A számok kerekítését a PHP, JavaScript, AS, és egyéb ECMA típusú nyelvekben is megtalálható ROUND(), FLOOR(), CEILING() függvényekkel, valamint TRUNCATE() és a FORMAT() függvényekkel végezhetjük el.

A ROUND(), FLOOR(), CEILING() függvények egyetlen számot várnak paraméterként. Ha a paraméter tizedes részének abszolút értéke  $\geq 0.5$  akkor a ROUND() az abszolút értékben nagyobb, különben az abszolút értékben kisebb, következő egészre kerekít.

A FLOOR() mindig a paraméternél kisebb, következő egész számot adja.

A CEILING() a FLOOR()-ral ellentétesen működik, azaz mindig a paraméternél nagyobb, következő egészet adja vissza.

A TRUNCATE() függvény két szám paramétert vár. Az első paraméterként kapott törtszámból, a második paraméterben megadott számú tizedes értéket hagy meg. Nem kerekít, a fölösleges tizedes jegyeket levágja. Ha a szám eleve kevesebb tizedes jeggyel rendelkezik, nem helyez el nullákat a megfelelő helyi értékeken.

```
SELECT PI () , TRUNCATE (PI () , 4) ;
+-----+-----+
| 3.141593 |      3.1415 |
+-----+-----+
```

A FORMAT(szám, tizedes\_jegy) eredményeként a szám #,###,###.## formátumban megadott számú tizedes jeggyel jelenik meg. Ha paraméter tizedeseinek száma kisebb, mint a tizedes jegyek várt száma, akkor a FORMAT nullákkal egészíti ki az eredményt. A FORMAT() egyébként a ROUND() függvénynek megfelelő kerekítést használja.



A számok előjelkezelésével kapcsolatba két függvény érdemel említést. A SIGN() függvény eredménye -1,0 vagy 1 attól függően, hogy a paraméterként kapott szám negatív, nulla, vagy pozitív. Az ABS() függvény egy a paraméter abszolút értékével tér vissza.

```
SIGN ( szám )
```

```
ABS ( szám )
```

### 5.5.3 Véletlenszámok generálása

Bár a MySQL biztosítja az erre megfelelő RAND() függvényt, az egyszerű véletlen számok előállítását nem túl gyakori adatbázis-kezelő feladat. Az azonban egyáltalán nem elképzelhetetlen, hogy véletlenszerű, de garantáltan egyedi azonosító létrehozására van szükségünk.

#### Véletlen számok

Egyszerű, véletlenszerűen generált számokat a RAND() függvénnyel hozhatunk létre.

A RAND() egyetlen egész típusú paraméterrel, és paraméter nélkül is hívható függvény. Visszatérési értéke 0 és 1 közötti tört szám ( $0 \leq \text{szám} < 1$ ).

Ha paraméterrel hívjuk, a véletlen szám generálásának alapja az átadott érték lesz.

Ha RAND() segítségével egész számokkal határolt intervallumba tartozó számokat akarunk generálni akkor a

```
FLOOR (@min + RAND () * (@max - @min+1))
```

kifejezés az @min..@max egész értékek közé eső számot generálja.

```
SET @min=10; SET @max=15;  
SELECT FLOOR(@min + RAND () * (@max - @min+1)) ;  
+-----+  
|                               15 |  
+-----+
```

#### Globálisan egyedi azonosító

Az UUID() függvény 32 darab 16-os számrendszerbeli számjegyekből álló, azaz 128 bites globális egyedi azonosítót (Universal Unique Identifier) hoz létre.

A xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx formátumú karaktersorozat számként nem használható, de kiválóan alkalmas egyedek globális azonosítására.

```
mysql> SELECT UUID();
+-----+
| f4aefdad-afe1-11e1-800b-74de2b031c72 |
+-----+
```

## 5.6 FÜGGVÉNYEK HASZNÁLATA A LEKÉRDEZÉSEKBEN

Mindeddig a függvények hívásával, funkcióival és formai leírásával ismerkedtünk meg. Itt az ideje, hogy lássunk néhány példát arra, hogy hogyan lehet őket választó lekérdezésekben is fölhasználni. Ehhez csak arra kell emlékeznünk, hogy a függvények paraméterként használhatjuk a lekérdezések forráshalmazának mezőit is.

Jelenítsük meg vásárlóink keresztneveit!

```
SELECT DISTINCT
  RIGHT(vNev, CHAR_LENGTH(vNev) - LOCATE(' ', vNev))
Kereszt
FROM vevo
ORDER BY Kereszt;
```

Kereszt
Beatrix
Berta
Bertalan
Boglárka
Domokos
Egyed
Emőd
Erika
Gáspár
Hermina
Huba
Ibolya
Izsó
Jácint
Judit
Lajos
Lehel
Lőrinc
Mária
Oszkár
Rebeka
Zsuzsanna

25. ábra Keresztnevek

Mutassuk meg a 2012-ben rendelést leadó vevők nevét és címét!

```
SELECT
    YEAR(rDatum) Ev,
    vNev,
    CONCAT_WS(", ",vIRSZ,vVaros, vUtca) Cim
FROM rendeles NATURAL JOIN vevo
WHERE YEAR(rDatum)=2012;
```

Ev	vNev	CONCAT_WS(", ",vIRSZ,vVaros, vUtca)
2012	Andrássy Mária	1011, Budapest, Kassa utca , 94
2012	Károly Jácint	3300, Eger, Szalag utca , 12
2012	Dirkó Rebeka	3600, Ózd, Bajnár utca , 97
2012	Miklósi Emőd	3501, Miskolc, Vári Szabó István utca , 24
2012	Andrássy Beatrix	3200, Gyöngyös, Szentgál utca , 21

26. ábra Vevők címei

Jelenítsük meg azokat a rendeléseket, amelyek leadása és kiszállítása között több, mint 8 nap telt el!

```
SELECT *,DATEDIFF(rSzallitonak,RDatum) Különbség  
FROM rendeles  
WHERE DATEDIFF(rSzallitonak,RDatum)>8;
```

## 5.7 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 5.7.1 Összefoglalás

Mai leckénkben a MySQL-ben használható függvények egy kicsiny részével, – mondhatni – töredékével ismerkedtünk meg. Megtanultuk, hogy a függvényeknek megfelelő beépített számításokat függvényhívás segítségével végeztethetjük el az adatbázis-kezelő rendszerrel.

A hívás helyén a függvény nevét, a kerek zárójelek közé írt, egymástól vesszővel elválasztott paramétereknek kell követniük. A paraméterek sorrendjét és számát a függvény specifikációja, formai leírása határozza meg.

Mivel a függvények visszatérési értékkel rendelkeznek, olyan helyen hívhatjuk őket, ahol értékükre valóban szükség van. A függvények visszatérési értéke fölhasználható számított mező kialakításakor, a WHERE záradék logikai kifejezéseiben és felhasználói változók értékadásában.



<http://dev.mysql.com/doc/refman/5.5/en/functions.html>

4. link: A MySQL függvényei és operátorai

### 5.7.2 Önellenőrző kérdések

1. Hol használhatunk függvényhívást?
  - Az SQL mondat olyan helyén ahol a visszatérési érték felhasználásra kerül. (Számított mező kifejezésében, logikai kifejezésben, értékadásban. )
2. Honnan tudjuk, hogy hány paramétert kell megadnunk egy függvény hívásakor?
  - Ezt a függvény specifikációja írja le.
3. Mi a különbség a CONCAT, és a CONCAT\_WS függvény között?
  - A CONCAT függvény egyszerűen összefűzi a paramétereit, a CONCAT\_WS hasonlóan dolgozik, de az összefűzött szövegek közé az első paraméternek megfelelő elválasztó szöveget tesz.
4. Hogyan tudná kiválasztani egy UUID középső 12 számjegyét?
  - `SELECT MID(UUID(),10,14);`
5. A rendeles tábla rDatum mezője tárolja a rendelés dátumát! Írjon lekérdézet, ami a 2011 augusztusában leadott rendelések összes mezőjét listázza!
  - `SELECT * FROM rendeles  
WHERE YEAR(rDatum)=2011 AND MONTH(rDatum)=8;`



# 6. LECKE: STATISZTIKAI SZÁMÍTÁSOK SQL-BEN

## 6.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Egy adatbázis táblái gyakran óriási mennyiségű rekordot tartalmaznak. A tárolt adatok megjelenítésekor általában nem az összes, csak a valamilyen feltételt kielégítő rekordokra vagyunk kíváncsiak. Éppen ezért a SELECT parancs megfelelő záradékaival, a kívánt információ tekintetében releváns rekordok kiválogatását írjuk elő. Az ilyenkor keletkező eredményhalmaz rekordjait ezután megvizsgálhatjuk, a megjelenített adatokból következtetéseket vonhatunk le, szükség esetén változtathatunk kiválogatás feltételein.

Ezek a lehetőségek igen hasznosak, sőt nélkülözhetetlenek, azonban nem adnak módot az eredményhalmazban megjelenő rekordok mezőértékeinek automatizált összevetésére.

Jelen tudásunkkal minden nehézség nélkül ki tudnánk válogatni a szükséges mezőket nyilvántartó táblából, a különböző településeken lakó vevőinket. Ha azonban az egyes településeken lakó vevők számát kellene megmondanunk, egyelőre csak manuális összegzéssel, verejtékes, és unalmas munkával tudnánk megoldani a feladatot.

A most következő lecke anyagának elsajátítása a több rekordot érintő összegző számítások automatizálásában nyit óriási távlatokat az olvasó előtt. A következő oldalak feldolgozásával megismerjük az összegző függvények fogalmát és használatukat. Megtanuljuk, hogyan érvényesíthetők összegzések egy választó lekérdezés összes rekordjára, és a rekordok szabályozott részhalmazaira.

A leckében törekedjen arra, hogy alaposan megértse az összegző függvények működését, és az összegzett részhalmazok kialakításának technikáját.

## 6.2 ÖSSZESÍTŐ LEKÉRDEZÉSEK

Az eddigiekben alkalmazott választó lekérdezések egy forráshalmazból mezők és rekordok kiválogatásával eredményhalmazt állítottak elő. A most következő, úgynevezett **összegző lekérdezések** valójában speciális választó lekérdezések. Specialitásuk abban rejlik, hogy egy választó lekérdezés segítségével kiválogatott rekordok mezőivel, **összegző műveleteket** végeznek el, és az eredményhalmazban ezek **eredményeit** jelenítik meg.

Az összesítő lekérdezések alábbi formai leírásának feltüntetése a lecke több további pontján is indokolt lenne. Ezért arra kérjük az olvasót, hogy a későbbi szakaszok olvasása közben, szükség esetén lapozzon vissza ide.

```
SELECT [részhalmaz_oszlopmeghat,][összesítő_számítás]
FROM rekordforrás
[GROUP részhalmaz_oszlopok][WITH ROLLUP]
[HAVING logikai_kifejezés]
[ORDER BY rendező_oszlopok];
```

Az alábbi ábrákon egy egyszerű választó lekérdezés majd egy összegző lekérdezés eredményhalmazát látjuk.

mysql> SELECT \* FROM vevő ORDER BY vUáros;

idUevő	vNév	vTelefon	vUáros	vUtca	vIRSZ
1	Andrássy Mária	(60) 832-4378	Budapest	Kassa utca , 94	1011
20	Ratkó Oszkár	(60) 434-3202	Budapest	Halászcserda utca , 4	1011
19	Kovács Lehel	(70) 909-4964	Eger	Thorma János utca , 13	3300
14	Kállai Gáspár	(20) 471-5961	Eger	Latabár utca , 1	3300
22	Kolonics Zsuzsanna	(60) 900-5824	Eger	Szilvás köz utca , 9	3300
11	Lénárd Domokos	(60) 638-9383	Eger	Mard utca , 82	3300
10	Liktor Bertalan	(20) 304-2231	Eger	Platán utca , 17	3300
24	Bereg Rebeka	(70) 825-3293	Eger	Peterka József utca , 8	3300
25	Simon Lőrinc	(20) 362-9654	Eger	Kolmann utca , 95	3300
28	Virga Izsó	(30) 339-2384	Eger	Uas utca , 74	3300
6	Károly Jácint	(70) 264-7074	Eger	Szalag utca , 12	3300
29	Keller Judit	(70) 626-6400	Eger	Kolmann utca , 47	3300
4	Károly Bertalan	(30) 891-6709	Eger	Cseresznye utca , 74	3300
3	Galam Huba	(60) 235-5859	Eger	Kolmann utca , 55	3300
17	Moka Ibolya	(70) 542-6072	Eger	Révész Gy. utca , 69	3300
23	Andrássy Beatrix	(30) 747-7979	Gyöngyös	Szentgál utca , 21	3200
2	Dallos Rebeka	(30) 584-4036	Gyöngyös	Mázsa utca , 5	3200
30	Hanga Berta	(30) 574-7970	Kazincbarcika	Kassa utca , 71	3700
8	Dévényi Jácint	(20) 696-4781	Kazincbarcika	Henger utca , 8	3700
9	Alapi Domokos	(30) 583-3755	Miskolc	Kovács Pál utca , 69	3501
5	Keller Berta	(20) 942-4648	Miskolc	Epreskert utca , 33	3501
21	Miklósi Enő	(70) 378-6676	Miskolc	Úári Szabó István utca , 24	3501
13	Pintye Lajos	(20) 737-7775	Miskolc	Szilvás köz utca , 30	3501
7	Pomogács Egyed	(60) 968-2916	Miskolc	Szilvás köz utca , 58	3501
16	Dirkó Rebeka	(60) 634-8517	Ózd	Bajnár utca , 97	3600
12	Szél Domokos	(70) 525-8777	Ózd	Kovács Pál utca , 4	3600
15	Szél Erika	(60) 756-3432	Pétervására	Venyige utca , 10	3250
18	Pelle Boglárka	(30) 695-5918	Pétervására	Dékáni Á. utca , 97	3250
26	Birtalan Beatrix	(30) 249-8345	Salgótarján	Henger utca , 61	3100
27	Andrássy Hernina	(60) 570-2956	Salgótarján	Híd utca , 10	3100

27. ábra Vevők felsorolása lakóhelyenként

Uáros	Uevők száma
Eger	13
Miskolc	5
Budapest	2
Pétervására	2
Ózd	2
Gyöngyös	2
Kazincbarcika	2
Salgótarján	2

28. ábra A vevők városonkénti számát megjelenítő lekérdezés



## 6.2.1 Összesítő függvények

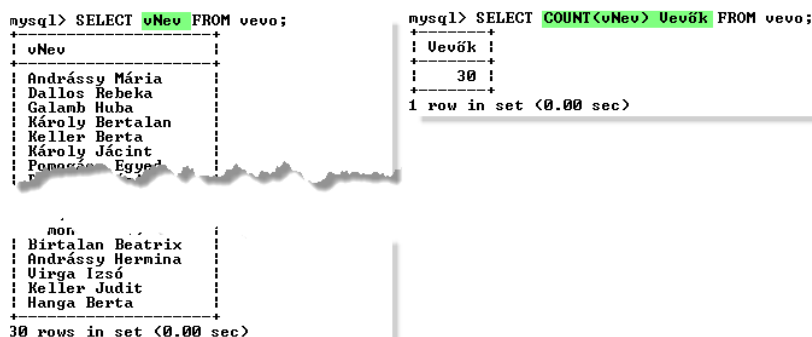
Mint tudjuk, adatainkat azért kell tárolnunk, hogy azok értelmezésére, az információ kinyerésére bármikor lehetőségünk legyen. Ahogyan arról már szoltunk, a SELECT parancsot követő mező meghatározások között kifejezéseket is elhelyezhetünk, azaz számított mezőket készíthetünk. Ez arra világít rá, hogy a relációs adatbázisok feldolgozásakor, nem csak a valóban tárolt mezőértékek, hanem az azokkal végzett számítások eredményeinek felhasználásával, értelmezésével is informálódhatunk. A számított mezők lehetőségeit tovább bővítették az előző leckében megismert függvények, amelyek speciális számításokat tesznek lehetővé.

Ezek függvények azonban csak elemi értékeket voltak képesek feldolgozni, azaz a híváskor átadott paramétereik mindegyike valamilyen egyszerű adattípus (dátum-idő, szöveg, szám...) lehetett. Az eddig tanult a függvények nem képesek tehát egy bizonyos paraméterben több értéket fogadni.

Az SQL nyelvben használható **összegző függvények** működése alapvetően ebben különbözik. Ezek a függvények nemcsak egy értéket, hanem **értékek sorozatát** kapják meg paraméterként. Megfogalmazhatnánk ezt úgy is, hogy nemcsak egy mezőérték, hanem egy mező több rekordban található értékeinek együttes feldolgozására képesek.

Minden összegző függvény egy mezőhivatkozást vár paraméterként. Működéskor a hivatkozott mező, adott rekordhalmazban előforduló összes értékével dolgozik.

Ilyen például a **COUNT (mező)** összegző függvény is, ami a paraméterben kapott mező egyes értékeinek megszámlálását végzi és a számítás eredményét adja vissza.



```
mysql> SELECT vNev FROM vevo;
+-----+
| vNev |
+-----+
| Andrásy Mária |
| Dallos Rebeka |
| Galamb Huba |
| Károly Bertalan |
| Keller Berta |
| Károly Jácint |
| Pongóczy Egyed |
| ... |
| Birtalan Beatrix |
| Andrásy Hermina |
| Virga Izsó |
| Keller Judit |
| Hanga Berta |
+-----+
30 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(vNev) Uevők FROM vevo;
+-----+
| Uevők |
+-----+
| 30 |
+-----+
1 row in set (0.00 sec)
```

29. ábra A COUNT függvény működése

A COUNT() paramétereként szokás mezőnév helyett csillagot (\*) megadni. Ez azt jelenti, hogy a függvény az összes mező kombinációját, azaz a rekordokat számolja meg.

## 6.3 A MYSQL ÖSSZESÍTŐ FÜGGVÉNYEI

A legfontosabb összesítő függvények az SQL összes nyelvjárásában megtalálhatók, azonban használatukban kisebb-nagyobb eltérések tapasztalhatunk.

Az alábbiakban a MySQL-ben használható összesítő függvényeket és működésüket soroljuk fel.

Függvény	Működés
AVG([DISTINCT] mező)	A paraméter értékeinek átlagát számolja
BIT_AND(mező)	A bit szintű AND műveletet végez a rekord-halmaz hivatkozott mezőjének értékein.
BIT_OR(mező)	A bit szintű OR műveletet végez a rekord-halmaz hivatkozott mezőjének értékein.
BIT_XOR(mező)	A bit szintű XOR műveletet végez a rekord-halmaz hivatkozott mezőjének értékein.
COUNT([DISTINCT] mező)	A paramétermező összes értékeit számolja meg
GROUP_CONCAT([DISTINCT] mező)	A paramétermező értékeit egyetlen karakterlánccá fűzi össze.
MAX([DISTINCT] mező)	A paramétermező értékei közül a legnagyobbat adja vissza
MIN([DISTINCT] mező)	A paramétermező értékei közül a legkisebbet adja vissza
STDDEV(mező)	A paramétermező értékeinek szórását számolja
STDEV_POP(mező)	A mezőértékeket a teljes populáció jellemzőjének tekintve végzi a szórás számítását.
STDEV_SAMP(mező)	A mezőértékeket egy populációból vett minta jellemzőjének tekintve becslést ad a populáció szórására.
SUM(mező)	A paraméter értékeinek összegét számolja
VARIANCE(mező)	A paraméter értékeinek varianciáját (szórásnégyzet) számolja ki.
VAR_POP(mező)	A mezőértékeket a teljes populáció jellemzőjének tekintve végzi a variancia számítását.
VAR_SAMP(mező)	A mezőértékeket egy populációból vett minta jellemzőjének tekintve becslést ad a populáció varianciájára.

A fenti táblázat néhány függvényében használható a DISTINCT kulcsszó. Alkalmazásakor a függvény, csak a paramétermező egymástól különböző értékeit használja fel az összegzésben. Elhagyásakor viszont a mező minden előforduló értékét feldolgozza.

A felsorolt függvények közül kiegészítő magyarázatot igényel a GROUP\_CONCAT(). A függvény a paramétermező értékeit szöveggé alakítja, majd ezeket (alapértelmezés szerint vesszővel elválasztva) egyetlen karakterláncá fűzi össze.

A függvény teljes specifikációja a következő:

```
GROUP_CONCAT ( [DISTINCT] mező
                [ORDER BY {mező| kifejezés}
                [ASC | DESC] [,mező...]]
                [SEPARATOR szöveg])
```

Mint látjuk, a függvény paramétereiként egyfajta beépített SQL-mondat adható meg, amelyben az ORDER BY záradék az összefűzés előtti rendezésre, a SEPARATOR pedig az elválasztó szöveg megadására való.



*Íme egy példa:*

*Mutassuk meg egyetlen szövegben a vevők lakóhelyeit úgy, hogy a városok neveit pontosvesszővel választjuk el!*



**SELECT GROUP\_CONCAT (**



**DISTINCT vVaros**



**ORDER BY vVaros**



**SEPARATOR ', ' ) Lista**



**FROM vevo;**

```

+-----+
| Lista |
+-----+
| Budapest; Eger; Gyöngyös; Kazincbarcika; Miskolc; Ózd; Pétervárasza; Salgótarján |
+-----+
1 row in set (0.00 sec)

```

30. ábra Városok neveiből alkotott szöveg

A leggyakrabban COUNT(), SUM(), AVG(), a MIN() és a MAX() összesítő függvényeket használjuk, ezért leckénk további részében is ezekre látunk majd példákat.

## 6.4 RÉSZHALMAZOK KIALAKÍTÁSA AZ ÖSSZESÍTŐ LEKÉRDEZÉSEKBEN

Részalmazok kialakítása nélkül az összesítő függvények a lekérdezés teljes eredményhalmazán elvégzik a paraméterként kapott mező összegzését. Az eredményhalmaz részekre bontása esetén azonban, a függvények részalmazonként, külön-külön összegzik paramétermező értékeit.

Az összegző lekérdezések részalmazait a GROUP BY záradékban megadott oszlop meghatározásokkal hozhatjuk létre. Az első oszlop meghatározás a teljes eredményhalmazt, a továbbiak pedig mindig az előző részalmazt bontják további részalmazokra. Az összesítő függvények mindig a legkisebb részalmazokon belül összegeznek.

Az összegző lekérdezés végső eredményhalmazát megtekintve láthatjuk, hogy az összegző függvényekben használt mezők összegzett eredménye jelenik meg. Ha a részalmazképző mezőket is látni szeretnénk, azok hivatkozásait a GROUP BY záradék mellett, a SELECT-et követő oszlop meghatározásokban is szerepeltetni kell.



*A következő lekérdezés például megmutatja, hogy hány vevő lakik az egyes városokban. A részalmaz képző mező a vVaros, az összegzett mező pedig a vNev:*



```
SELECT vVaros, COUNT(vNev) Vevők
FROM vevo
GROUP BY vVaros;
```

```
mysql> SELECT vUaros, COUNT(vNev) Vevők FROM vevő GROUP BY vUaros;
```

vUaros	Vevők
Budapest	2
Eger	13
Gyöngyös	2
Kazincbarcika	2
Miskolc	5
Ózd	2
Pétervására	2
Salgótarján	2

```
8 rows in set (0.00 sec)
```

31. ábra Vevők számának összegzése városonként

Fontos tudni, hogy részhalmazokat nemcsak mezők, hanem számított értékek alapján is képezhetünk.



Ez a példa a rendelések számát mutatja meg év és hónap szerinti bontásban. A rendeles tábla rekordjaiból a `YEAR(rDatum)` kifejezés alapján hozzuk létre a nagyobb részhalmazokat, majd ezeket a `MONTHNAME(rDatum)` kifejezéssel további részhalmazokra bontjuk. Az egyes hónapokon belül megszámloljuk a rendeléseket.



**SELECT**



**YEAR(rDatum) Év,**



**MONTHNAME(rDatum) Hó,**



**COUNT(\*)**



**FROM rendeles**



**GROUP BY YEAR(rDatum), MONTHNAME(rDatum)**



**ORDER BY Év, MONTH(rDatum);**

Év	Hó	COUNT(*)
2010	January	1
2010	March	3
2010	April	1
2010	June	1
2010	July	1
2010	August	3
2010	October	2
2010	November	1
2010	December	4
2011	January	3
2011	February	1
2011	March	1
2011	April	1
2011	June	1
2011	July	1
2011	August	2
2011	September	2
2011	November	1
2012	February	1
2012	March	2
2012	April	2

21 rows in set (0.00 sec)

32. ábra Összegzés részhalmazokkal

Az összegző lekérdezések készítése álnevek használatával egyszerűsíthető. Ha a SELECT-et követő mezőmeghatározásokban álneveket adunk a később részhalmaz képzésre használt mezőknek, akkor a GROUP BY záradékban már az álnevekkel hivatkozhatunk az oszlopokra.

Ez a lekérdezés az előzővel azonos eredményt ad:

```

SELECT
    YEAR(rDatum) Év,
    MONTHNAME(rDatum) Hó,
    COUNT(*)

FROM rendeles

GROUP BY Év, Hó

ORDER BY Év, MONTH(rDatum);

```

## 6.5 REKORDOK KIVÁLOGATÁSA

A SELECT parancs kapcsán megismertük a WHERE záradékot, amellyel az eredményhalmazba kerülő rekordok kiválogatásának feltételei adhatók meg. A WHERE záradék az összesítő lekérdezésekben is használható, amennyiben a feltétel a forráshalmaz mezőire vonatkozik. Ha azonban a számított érték alapján akarjuk kiválogatni a rekordokat, akkor WHERE helyett a HAVING záradékot kell használnunk a feltételek megadásakor. A HAVING záradékban a WHERE-hez hasonlóan egyszerű, vagy összetett logikai kifejezésekkel szabályozhatjuk a lekérdezésben szereplő rekordokat.



*Az alábbi példában csak azokat az hónapokat mutatjuk meg, ahol a rendelések száma legalább 3 volt:*



**SELECT**



**YEAR(rDatum) Év,**



**MONTHNAME(rDatum) Hó,**



**COUNT(\*) Rendelés**



**FROM rendeles**



**GROUP BY Év, Hó**



**HAVING Rendelés>=3**



**ORDER BY Év, MONTH(rDatum);**

Év	Hó	Rendelés
2010	March	3
2010	August	3
2010	December	4
2011	January	3

4 rows in set (0.00 sec)

33. ábra Legalább 3 rendelés

Talán feltűnt az olvasónak, hogy a WHERE záradékban sohasem hivatkozunk álnevekkel a mezőkre, még az ORDER BY és a most bemutatott HAVING záradékban igen. Ennek oka az eredményhalmaz kialakításának lépéseiben keresendő.

Amikor a DBMS feldolgoz egy SQL-mondatot, előbb kiértékeli a rekordforrást, és létrehozza forráshalmazt. Ezt követően a WHERE záradék feldolgozásával kiválogatja a megfelelő rekordokat. Csak ezek után dolgozza föl a SELECT-et követő mező meghatározásokat. Mivel a WHERE záradék értelmezése előbb következik be, itt még nem hivatkozhatunk a később kialakításra kerülő álnevekre.

Miután az oszlopmeghatározásokkal végzett, a DBMS földolgozza a többi záradékot is, ahol már használhatjuk a mezők álneveit.

## 6.6 WITH ROLLUP MÓDOSÍTÓ

Az összesítő lekérdezések érdekes lehetősége a GROUP BY záradék WITH ROLLUP módosítója. Ha a módosítót feltűntetjük a záradék végén, akkor az eredményhalmazban az összes részhalmaz megjelenése után megkapjuk a részhalmazok egészére vonatkozó összegzést is.



**SELECT**



**YEAR(rDatum) Év,**



**MONTHNAME(rDatum) Hó,**



**COUNT(\*) Rendelés**





**FROM rendeles**



**GROUP BY Év, Hó WITH ROLLUP;**

Év	Hó	Rendelés
2010	April	1
2010	August	3
2010	December	4
2010	January	1
2010	July	1
2010	June	1
2010	March	3
2010	November	1
2010	October	2
2010	NULL	17
2011	April	1
2011	August	2
2011	February	1
2011	January	3
2011	July	1
2011	June	1
2011	March	1
2011	November	1
2011	September	2
2011	NULL	13
2012	April	2
2012	February	1
2012	March	2
2012	NULL	5
NULL	NULL	35

34. ábra WITH ROLLUP módosító alkalmazása

A WITH ROLLUP módosító használata esetén az eredményhalmaz a részhalmaz képző mezők szerint rendezett. Az ORDER BY záradék ilyenkor nem használható.

## 6.7 ÖSSZESÍTŐ LEKÉRDEZÉSEK GYAKORLATI ALKALMAZÁSA

Most befejezett leckénk új ismeretanyaga jó példa az adatbázis-kezelésben – de általában az informatikában – jellemző jelenségre, amikor viszonylag kevés új ismeret nyitja meg az alkalmazás meglehetősen széles horizontját. Lássunk most néhány példát az összegző lekérdezések alkalmazási lehetőségeire a **webbolt** adatbázis használatával!



*Határozzuk meg a nyilvántartott termékek számát!*



***SELECT COUNT(\*) FROM termék;***



*Mutassuk meg a termékek kategóriánkénti számát!*



***SELECT k.kNev Kategória, COUNT(t.idTermek)  
Termékszám***



***FROM***



***termék t NATURAL JOIN kategória k***



***GROUP BY k.kNev;***



*Mekkora bevételre tettünk szert az egyes években?*



***Select YEAR(r.rDatum) Év,  
SUM(t.tEladAr\*rt.DarabSzam) Összeg***



***From***



***rendeles r***



***INNER JOIN rendelestermek rt***



***ON rt.idRendeles = r.idRendeles***



***INNER JOIN***



*termek t*



*ON rt.idTermek = t.idTermek*



*GROUP BY Év*



*Mennyi volt az egyes évek átlagos bevétele?*



*Select YEAR(r.rDatum) Év,  
format (AVG(t.tEladAr\*rt.DarabSzam),2) Összeg*



*From*



*rendeles r*



*INNER JOIN rendelestermek rt*



*ON rt.idRendeles = r.idRendeles*



*INNER JOIN*



*termek t*



*ON rt.idTermek = t.idTermek*



*GROUP BY Év*



*Melyik a három legnépszerűbb termékünk?*



```
Select t.tMegnevezes Termék,  
SUM(rt.DarabSzam) Eladva
```



```
From
```



```
rendelestermek rt
```



```
INNER JOIN
```



```
termek t
```



```
ON rt.idTermek = t.idTermek
```



```
GROUP BY Termék
```



```
ORDER BY Eladva DESC
```



```
LIMIT 3
```



Mennyibe kerülnek az egyes termékkategóriák legolcsóbb termékei?



```
SELECT MIN(t.tEladAr), k.kNev
```



```
FROM
```



```
termek t INNER JOIN categoria k
```



```
ON t.idKategoria = k.idKategoria
```



**GROUP BY k.kNev**



*Mennyi az egyes termékkategóriák átlagos árszintje?*



**SELECT       FORMAT (AVG (t.tEladAr) ,2)       Átlagár,  
k.kNev**



**FROM termék t INNER JOIN**



**kategoria k ON t.idKategoria =  
k.idKategoria**



**GROUP BY k.kNev**

## 6.8 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 6.8.1 Összefoglalás

Az összesítő lekérdezésekben egy rekordcsoport kiválasztott mezőjének minden értékét feldolgozó, összesítő függvényekkel végezhetünk statisztikai műveleteket. A rekordcsoport lehet egy választó lekérdezés eredményhalmazának összes rekordja, vagy az azokból képzett részhalmazok egyike.

Az összesítő lekérdezések elkészítésekor az összesítő függvényekkel készített számított mezőket mindenképpen fel kell sorolni a lekérdezés mező meghatározásai között. A részhalmazképző mezők felsorolására csak akkor van szükség, ha meg akarjuk azokat jeleníteni az eredményhalmazban. A részhalmazok kialakítását a GROUP BY záradékkal írhatjuk le, a részhalmazképző mezők, vagy álneveik vesszővel elválasztott felsorolásával. A részhalmazképzés szintjeit a GROUP BY záradékot követő mezők és sorrendjük határozza meg.

A releváns rekordok kiválasztására itt is használható a WHERE záradék, de ha a feltétel valamelyik összesítés eredményére vonatkozik, akkor a HAVING záradékot kell alkalmazni.

A GROUP BY záradék kiegészítő lehetősége a WITH ROLLUP módosító, aminek alkalmazásával a részhalmazok összegzésével kiszámítható érték is megjelenik.

### 6.8.2 Önellenőrző kérdések

1. Miben különböznek az összesítő lekérdezések a választó lekérdezésektől?
  - Az összegző lekérdezések választó lekérdezések eredményhalmazát dolgozzák fel úgy, hogy azt részhalmazokra bontják, és a részhalmazok egyes mezőin összegző műveleteket hajtanak végre.
2. Melyik összesítő függvényekkel lehet egy mező értékeinek számát, a legkisebb, legnagyobb, és az átlagértéket meghatározni?
  - COUNT(), MIN(), MAX(), AVG()
3. Mi a jelentősége az egyes összesítő függvényekben használható DISTINCT kulcsszónak?
  - Ha nem használjuk a függvény a paraméterként kapott mező rekordhalmazon belüli összes értékét feldolgozza. Ha használjuk, akkor csak az egymástól különböző értékekkel végez műveletet.
4. Lehet-e egy lekérdezésben többféle összegzést végezni?
  - Igen, de csak azonos részhalmazokon.
5. Miben különbözik a WHERE és a HAVING használata az összesítő lekérdezésekben?
  - A WHERE feltételeit még az összegzés előtt, a HAVING-et az összegzés után dolgozza fel a DBMS. Ezért a WHERE csak a nem összegezett mezőkre vonatkozhat, a HAVING pedig az összegzések eredményére.

## 7. LECKE: ALLEKÉRDEZÉSEK HASZNÁLATA

### 7.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

Az adatbázisok lekérdezésével kapcsolatos munkánk során előbb, vagy utóbb, de biztosan eljutunk olyan feladatokhoz, amelyek első pillantásra egyszerűnek tűnnek, alaposabban átgondolva azonban egyetlen lekérdezés alkalmazásával megoldhatatlannak bizonyulnak. Ilyen eset adódik akkor, amikor egy lekérdezés valamilyen elemét csak egy másik lekérdezéssel tudjuk előállítani. Lássunk erre egy konkrét példát!

Kik azok a munkatársaink, akik az átlagosnál kevesebb megrendelést dolgoztak fel?

A lekérdezés nem tűnik bonyolultnak. A **munkatars-rendeles** tábla kapcsolata alapján meg kell határoznunk a munkatársanként feldolgozott megrendelések számát, majd kiválogatni azokat, akiknél ez az érték az átlag alatt van. A problémát az utolsó feltétel okozza. Az átlagos megrendelés számot ugyanis csak lekérdezéssel, mégpedig egy összegző lekérdezéssel tudjuk előállítani. Természetesen megtehetjük, hogy előbb kiszámoljuk a feldolgozott megrendelések átlagos számát, majd ezt az értéket a következő lekérdezésbe helyettesítve, kiválogatjuk a megfelelő munkatársakat. Túl azon, hogy a megoldás nem kifejezetten elegáns, hibalehetőséget is hordoz magában. Az első és második lekérdezés futtatása közben ugyanis, akár jelentős idő is eltelhet. Közben az adatbázis állapota megváltozhat (újabb rendeléseket dolgoznak fel) így második lekérdezés nem ad biztosan pontos eredményt.

Az ilyen feladatok megoldása biztonságosabb, és „tisztább” is, ha a lekérdezésünkbe egy másik, úgynevezett allekérdezést építünk, amely a fő lekérdezés számra szolgáltat adatokat.

Most következő leckénkben azt fogjuk áttekinteni, hogyan készíthetjük el, és mi mindenre használhatjuk az allekérdezéseket. Elsőként megtanuljuk az allekérdezések összeállításának formai és logikai követelményeit. Ezután megvizsgálunk néhány, csak allekérdezéssel megoldható problémát, illetve pár olyan feladatot, amelyben az allekérdezések használata alternatív lehetőséget kínál.

Most következő leckénk ismeretei rendkívül sok manuális munkától kímélhetik meg az adatbázis kezelőjét. Ez már leckénk példái alapján is beláthatóvá válik, de a későbbi fejezetekben arról is meggyőződhetünk, hogy ezek az eszkö-

zök nemcsak a rekordokat kiválasztó DQL, hanem a rekordok törlésre, változtatásra, és beszúrára használható DML, mi több, az adatdefiniáló DDL résznyelvben is használhatók.

Kérjük tehát az olvasót, hogy ne csupán elolvassa és értelmezze a most következő leckét, de alaposan vizsgálja meg a bemutatott példákat, majd próbáljon maga is allekérdezésekkel megoldható feladatokat megfogalmazni!

A leckében keressen válaszokat az alábbi kérdésekre:

- Mit jelentenek pontosan az allekérdezés és főlekérdezés fogalmak?
- Milyen formában illeszthetők a főlekérdezésbe az allekérdezések?
- A lekérdezések mely pontjain használhatunk allekérdezést?
- Hogyan használhatók az allekérdezések feldolgozásakor az IN, ANY, ALL, illetve EXISTS, NOT EXISTS kulcsszavak?

## 7.2 FŐ- ÉS ALLEKÉRDEZÉS

Előfordulhat, hogy egy SQL-mondat valamely eleme, csak egy másik SQL-mondat eredményhalmaza, vagy annak részeként állítható elő. Az ilyen feladatok megoldására használt SQL-mondat valójában két, egymásba ágyazott lekérdezéssel fogalmazható meg. A megoldás nagyon hasonlít a beszélt nyelvek összetett mondataihoz, amelyek tagmondatokból állnak. Az „összetett” SQL-mondatok esetében azonban, meghatározzuk a tagmondatok hierarchiáját, és fő- valamint allekérdezésekről beszélünk. Az **allekérdezés a beágyazott**, a **főlekérdezés a befoglaló** lekérdezés. Másképpen fogalmazva a főlekérdezés az, amelynek valamelyik nyelvi elemét egy beágyazott lekérdezés, az allekérdezés helyettesíti.



*Melyek azok a termékek, amelyek olcsóbbak az átlagnál?*



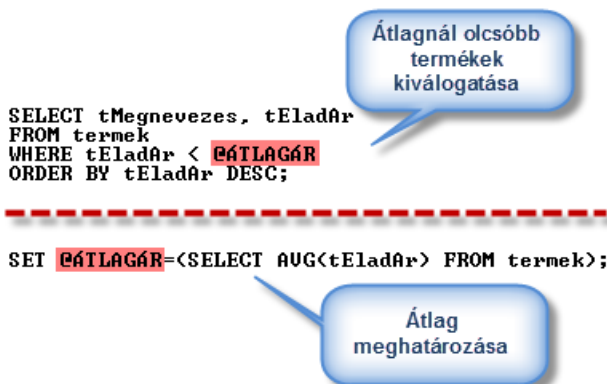
*A főlekérdezés termékeket válogat ki, mégpedig olyanokat, amelyek ára kevesebb, mint az átlagos termékár. Az átlagos ár azonban csak egy összesítő lekérdezéssel határozható meg. Az allekérdezés ezt a feladatot láthatja el.*

Az esetek többségében az allekérdezés szolgáltat valamilyen adatot a főlekérdezés számára. Léteznek olyan esetek is, amikor az allekérdezés a főlekérdezés valamely mezőértékének felhasználásával működik.



## 7.3 ALLEKÉRDEZÉS BEILLESZTÉSE

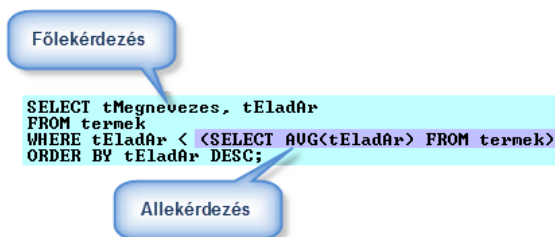
Az átlagnál olcsóbb termékeket megjelenítő lekérdezés allekérdezés nélkül az alábbi ábrán, a szaggatott vonal fölött látható formában képzelhető el. Ahhoz, hogy valóban működjön, szükség van az átlagos árat tartalmazó **@ÁTLAGÁR** felhasználó változóra, aminek értékét a vonal alatt látható lekérdezéssel számolhatjuk ki.



35. ábra Átlagnál olcsóbb termékek

Ha valóban így szeretnénk használni a lekérdezéseket, akkor előbb a változó értékét kellene meghatározni, csak azután indíthatnánk el a termékeket kiválogató lekérdezést.

A fenti feladat egyetlen összetett lekérdezéssel is megoldható úgy, hogy az **@ÁTLAGÁR** változó értékét kiszámoló lekérdezést beillesztjük a termékeket kiválogató lekérdezésben, a változó helyére.



36. ábra Fő- és allekérdezés

A fenti példában az allekérdezés a főlekérdezés **WHERE** záradékában megadott logikai kifejezésben található. Figyeljük meg, hogy az allekérdezés kerek zárójelek (...) között található, és egyetlen értéket, egy számot szolgáltat.

**A főlekérdezésbe ágyazott allekérdezéseket mindig kerek zárójelek között kell elhelyezni!**

**Az allekérdezésnek annyi mezőt, és rekordot kell szolgáltatnia, amennyire a főlekérdezésben a beágyazás helyén szükség van!**

A fent megfogalmazott első szabálynak nem nehéz megfelelni, a másodikra azonban különösen oda kell figyelni.

Az allekérdezések ugyanis a visszaadott eredmény szempontjából négy csoportba sorolhatók:

- A **skalár** allekérdezések egyetlen értéket állítanak elő.
- A **mező**, vagy **oszlop** allekérdezés egy teljes mezőt, értékek oszlopát adja át a főlekérdezésnek.
- A **sor**, vagy **rekord** allekérdezés egy teljes rekordot ad vissza.
- A **tábla** allekérdezés rekordsorozatot, táblát ad eredményül.

Allekérdezés készítésekor jól gondoljuk meg, hogy a beágyazás helyére skalárnak, mezőnek, rekordnak, vagy táblának kell-e kerülnie. Az allekérdezést ennek megfelelően kell kialakítanunk!

## 7.4 ALLEKÉRDEZÉSEK HELYE A FŐLEKÉRDEZÉSBEN

Allekérdezést a választó lekérdezés típusú főlekérdezések alábbi elemeiben elhelyezni:

- SQL-mondat mezőmeghatározásai között, egyfajta számított értéként,
- SQL-mondat rekordforrásában, annak elemeként,
- SQL-mondat WHERE záradékában, logikai kifejezés tényezőjeként.

A következő leckékben látni fogjuk, hogy az allekérdezések a DML-lekérdezések elemeiként is használhatók.

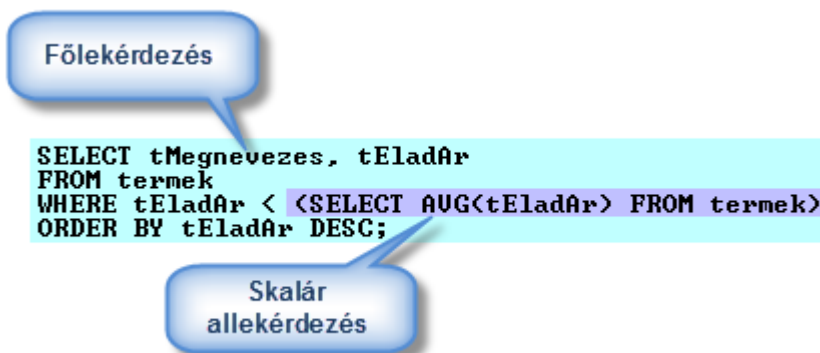
### 7.4.1 Allekérdezés WHERE záradékban

Bár tananyagunk minden felhasználási típusára mutat példát, nyugodtan kijelenthetjük, hogy az allekérdezéseket elsősorban a főlekérdezések **WHERE** záradékában, a rekordok kiválogatásának feltételeként használjuk.

Ezekben az esetekben általában főlekérdezés a forráshalmazának egy mezőjét, az allekérdezés által visszaadott egyetlen értékkel hasonlítjuk össze. Mivel allekérdezésnek annyi mezőt és rekordot kell szolgáltatnia, amennyire a főlekérdezésben szükség van, **az ilyen esetek zömében skalár allekérdezést kell készítenünk**. Előfordulhat azonban, hogy a feladat mégis mező típusú allekérdezést kíván meg.

#### Skalár típusú allekérdezések használata

Skalár típusú allekérdezés használatára láttunk példát akkor, amikor az átlagos árnál olcsóbb termékek megjelenítését végző lekérdezést tanulmányoztuk. A példában a főlekérdezés forráshalmazának **egy mezőjét**, az allekérdezés által visszaadott **értékkel hasonlítottuk** össze. Pontosan ezért skalár allekérdezésre volt szükség.



37. ábra Skalár allekérdezés WHERE záradékban

#### Mező típusú allekérdezések használata

A következő példában az allekérdezés egy mezőt (mezőértékek sorozatát, ha úgy tetszik, egy mezőből álló táblát) ad vissza



*Kik azok a vevők, akik cégünk bármelyik munkatársával azonos városban (településen) laknak?*



*A feladat megoldásához használt főlekérdezés kiválogatja a meghatározott városokban lakó vevőket. A lehetséges városok neveit azonban egy a munkatársak lakóhelyeit lekérdező allekérdezés fogja összeállítani.*



**SELECT vNev, vVaros**



**FROM vevo**



**WHERE vVaros IN (SELECT mVaros FROM munkatars);**

A fenti példában mező típusú allekérdezést készítünk, amely több rekordot, de mindben csak egy mezőt tartalmaz. Valójában egy az **mVaros** mező értékeiből álló halmazt kapunk, majd a főlekérdezésben az **IN** operátorral vizsgáljuk, hogy a megfelelő mező (**vVaros**) aktuális értéke eleme-e ennek a halmaznak.

**Ha egy allekérdezés mező típusú, akkor az **IN** operátor alkalmas az annak a főlekérdezés egyetlen mezőjével való összehasonlításra.**

A feladat fenti megoldása a FROM záradékban megfogalmazható kapcsolás (JOIN) egy alternatívája lehet. Lássuk, hogyan készíthettük volna el a lekérdezést JOIN segítségével:

```
SELECT DISTINCT v.vNev, v.vVaros
FROM vevo v INNER JOIN munkatars m
ON v.vVaros=m.mVaros
```

Mint láttuk az **IN** operátor alkalmas arra, hogy a **WHERE** záradékban egy mező értékeit, halmazzal, azaz mező típusú allekérdezés eredményével hasonlítsuk össze.

Az allekérdezések **WHERE** záradékba illesztésekor azonban használhatjuk még az **ANY**, és az **ALL** kulcsszavakat is, amelyekkel más operátorokkal (<>!=...) is működnek.

**WHERE**

*főlekérdezés mezője*  
operátor **ANY | ALL**  
(*mező\_típusú\_allekérdezés*)

A *főlekérdezés\_mezője* az *operátor* és a *mező\_típusú\_allekérdezés* valójában logikai kifejezést alkot. Ez az ANY esetén akkor igaz, ha az allekérdezés eredményhalmazának **bármelyik** mezőértékére igaz. Az 'ALL' esetén pedig akkor, ha az allekérdezés **minden** egyes mezőértékére igaz.



A következő lekérdezés minden olyan terméket megjelenít, amely szerepel valamelyik megrendelésen. Azaz már eladtunk belőle legalább egyet.



**SELECT idTermek, tMagnevezes**



**FROM termek**



**WHERE idTermek = ANY (**



**SELECT idTermek FROM rendelestermek**



**)**



Az operátor kulcsszó és megváltoztatásával megfordítjuk az eredményt. Azokat termékeket listázzuk, amelyeket még sosem rendeltek tőlünk:



**SELECT idTermek, tMagnevezes**



**FROM termek**



**WHERE idTermek != ALL (**



**SELECT idTermek FROM rendelestermek**



**)**



Ha csak az **ANY**-t cseréltük volna **ALL**-ra, üres halmazt kaptunk volna. Hisz ez azt jelentené, hogy olyan terméket akarunk kiválasztani a **termek** táblából, aminek **idTermek** mezője az allekérdezés minden **idTermek** mezőjével egyenlő. (Ez csak akkor fordulhatna elő, ha a termék táblában csak egy rekord lenne.)



Mi azonban az egyenlő operátort (=) nem egyenlőre (!=) cseréltük. Az = **ANY** azt jelenti, legalább az egyikkel egyenlő, != **ALL** azt, hogy „mindegyikkel nem egyenlő”, azaz egyikkel sem egyenlő. Így azokat a termékeket jelenítjük meg, amelyek azonosítója a legalább egyszer eladott termékek azonosítóit kiválogató allekérdezés eredményének egyik értékével sem egyenlő.

Egyes feladatokban a főlekérdezés WHERE záradékában csupán arra vagyunk kíváncsiak, hogy az allekérdezés egyáltalán tartalmaz-e rekordot. Ilyenkor a WHERE záradék logikai kifejezésében csupán az EXISTS (létezik) vagy az NOT EXISTS (nem létezik) kulcsszavak egyike, és az allekérdezés szerepel.



Az előző két feladat megoldható lett volna akkor is, ha az allekérdezés kiválogatta volna **rendelesstermek** táblából azokat a rekordokat, amelyekben szerepel a főlekérdezés aktuális **idTermek** értéke. Ezt követően a főlekérdezésben már csak azt kellett volna vizsgálni, hogy van-e egyáltalán rekord az allekérdezés eredményhalmazában, vagy üres halmazról van szó.



Figyeljük meg, hogy a feladat megoldásában a tábláknak adott álnevek használatával oldjuk meg a többértelmű mezőhivatkozások kiküszöbölését:



```
SELECT idTermek, tMegnevezes
```



```
FROM termék t
```



```
WHERE NOT EXISTS
```



```
(
```



```
SELECT idTermek FROM rendelestermek rt
```



```
WHERE rt.idTermek=t.idTermek
```



```
)
```

### 7.4.2 Allekérdezés a rekordforrásban

Vannak esetek, amikor egy allekérdezés eredményhalmazára a **főlekérdezés forráshalmazának meghatározásakor** van szükség. Ilyenkor a főlekérdezés rekordforrásában használjuk az allekérdezéseket.

Példafeladatunk legyen a következő:



*Mutassuk meg, kik azok a vevők, akik abban a városban laknak, ahonnan a legkevesebb vevőnk van?*



*A kissé nyakatekert feladatmegfogalmazást értelmezve hamar rájövünk arra, hogy a most elkészítendő főlekérdezésnek a **vevo** táblán kell alapulnia. A táblából azokat a rekordokat kell kiválogatni, amelyekben a **vVaros** mező értéke egy bizonyos város.*



```
SELECT vNev, vVaros
```



**from vevo**



**WHERE vVaros=@Kevésvevő**



**;**



Most a szükséges városnevet kell allekérdezéssel meghatároznunk. Az alábbi SQL mondat lakóhelyek szerint sorolja részhalmazokba a **vevo** tábla rekordjait, és megszámlolja az egyes részhalmazok sorait. Végeredményben meghatározza, hogy az egyes városokban hány vevő lakik. A Rekordokat városenkénti vevőszám alapján növekvő sorrendbe állítja, és csak az első rekordot adja vissza. Eredményként tehát egy rekordot kapunk, amelynek **vVaros** mezője a megfelelő város nevét, **Vevők** mezője pedig az ott lakó vevőink számát tartalmazza:



**SELECT vVaros,COUNT(\*) Vevők**



**FROM vevo**



**GROUP BY vVaros**



**ORDER BY Vevők**



**LIMIT 1;**



+-----+-----+






| **vVaros** | **Vevők** |



+-----+-----+



 | Pétervására | 2 | +-----+ 1 row in set (0.00 sec) Az eddig tanultak felületes olvasása esetén kínálkozna az alábbi megoldás: *SELECT vNev, vVaros* *from vevo* *WHERE vVaros=(* *SELECT vVaros,COUNT(\*) Vevők* *FROM vevo* *GROUP BY vVaros* *ORDER BY Vevők* *LIMIT 1* *)* *;*



*Az allekérdezés ilyen használata azonban hibához vezetne, mert a főlekérdezésben egy mezőt várunk, az allekérdezéssel viszont kettőt adunk vissza. Az alkalmazott allekérdezés mező meghatározásából pedig nem hagyható el, a **COUNT(\*) Vevők** oszlop meghatározás, mert az összegzett mezőnek mindenképpen szerepelni kell az összesítő lekérdezésben.*



*A megoldás tehát az, hogy az allekérdezést rekordhalmaznak tekintve, a főlekérdezés rekordforrásának megadásánál, a FROM záradékban használjuk.*



**SELECT v1.vNev, v1.vVaros**



**FROM**



**vevo v1**



**INNER JOIN**



**(**



**SELECT vVaros, COUNT(\*) Vevők**



**FROM vevo**



**GROUP BY vVaros**



**ORDER BY Vevők desc**



**LIMIT 1**



) v2



ON v1.vVaros=v2.vVaros



;



*Figyeljük meg, hogy a rekordforrásban lévő **vevo** tábla, és az allekérdezés is álnevet kap (v1, v2). Ez arra ad lehetőséget, hogy a főlekérdezésben egyértelműen tudjunk hivatkozni az allekérdezés mezőire.*



*Ebben a megoldásban a főlekérdezés rekordforrásában lévő **vevo** táblát szoros illesztéssel (INNER JOIN) kapcsoltuk az allekérdezéshez, majd a rekordkapcsolatok kialakításának feltételként az **ON v1.vVaros=v2.vVaros** kifejezés adtuk meg. Így a lekérdezés forráshalmaza már eleve a megfelelő vevőket tartalmazza.*



+-----+-----+



| vNev | vVaros |



+-----+-----+



| Galamb Huba | Eger |



| Károly Bertalan | Eger |



| Károly Jácint | Eger |




| Lektor Bertalan | Eger |

		<i>Lénárd Domokos</i>		<i>Eger</i>	
		<i>Kállai Gáspár</i>		<i>Eger</i>	
		<i>Moka Ibolya</i>		<i>Eger</i>	
		<i>Kovács Lehel</i>		<i>Eger</i>	
		<i>Kolonics Zsuzsanna</i>		<i>Eger</i>	
		<i>Bereg Rebeka</i>		<i>Eger</i>	
		<i>Simon Lőrinc</i>		<i>Eger</i>	
		<i>Virga Izsó</i>		<i>Eger</i>	
		<i>Keller Berta</i>		<i>Eger</i>	
	+-----+-----+				
	<i>13 rows in set (0.00 sec)</i>				

### 7.4.3 Allekérdezések mező meghatározásban

Előfordulhat, hogy egy allekérdezés eredményére már a főlekérdezés mezőmeghatározásai között szükség van.

 *Tegyük föl, hogy látni szeretnénk a forgalmazott termékeink neveit, eladási áraikat, és azok az átlagos ártól való eltérését.*



A feladat megoldásához a **termek** tábla rekordjaiból meg kell jelenítenünk a **tMegnevezes**, és **tEladAr** mezőket, de használnunk kell egy számított mezőt is, amely az **tEladAr** és az átlagos ár közötti különbséget tartalmazza. Ezzel a mezővel azonban az a probléma, hogy – mint azt előző feladatban láttuk – értéke csak allekérdezéssel számolható ki. A megoldás tehát a következő:



**SELECT**



**tMegnevezes,**



**tEladAr,**



**tEladAr - (SELECT AVG(tEladAr) FROM termék)**  
**Különbség**



**FROM termék**



**ORDER BY tMegnevezes;**



Az allekérdezés a **Különbség** álnevű számított mezőt meghatározó kifejezés egy tényezője. A kivonásban elfoglalt helyén egyetlen értékre van szükség, ezért itt is **skalár allekérdezés kell használnunk**. A példában használt **(SELECT AVG(tEladAr) FROM termék)** megfelel ennek a feltételnek, hiszen az eredménye egyetlen érték, az átlagár.

## 7.5 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 7.5.1 Összefoglalás

Ebben a leckében az allekérdezések használatával foglalkoztunk. Megismertük a fő- és allekérdezés fogalmát, megtanultuk az allekérdezések beágyazásának nyelvtani formáját, és logikai feltételét. Már tudjuk, hogy az

allekérdezés mindig kerek zárójelek között kerül a főlekérdezés megfelelő helyére, és hogy annyi rekordot és mezőt kell szolgáltatnia, amennyit a beágyazás helyén a főlekérdezés igényel. Ezzel kapcsolatban megismertük a skalár, mező, rekord, és tábla allekérdezések közötti különbséget.

Áttekintettük az allekérdezések választó lekérdezésekbe ágyazásának lehetséges helyeit. Láttuk, hogy az oszlop meghatározásokban, a rekordforrás megadásában és a WHERE záradék logikai kifejezésében egyaránt alkalmazhatók allekérdezések. Megismertük az IN, ANY, ALL, EXISTS és NOT EXISTS kulcsszavakat, amelyekkel biztosítható a mező és a tábla típusú allekérdezések WHERE záradékban történő használata.

### 7.5.2 Önellenőrző kérdések

1. Mikor érdemes allekérdezést használni?
  - Allekérdezést akkor érdemes használni, amikor a főlekérdezés valamilyen skalár, rekord, mező, vagy tábla elemét csak egy másik lekérdezéssel lehet előállítani.
2. Mit kell tudni az allekérdezés beágyazásának nyelvtani és logikai feltevéleről?
  - Az allekérdezéseket kerek zárójelek között illesztjük a főlekérdezésbe. Sosem zárjuk őket pontosvesszővel, hiszen ez a karakter a főlekérdezés végét jelzi. Az allekérdezés által visszaadott mező és rekordszám meg kell feleljen a főlekérdezés elvárásainak.
3. Mit értünk skalár, rekord, mező illetve tábla allekérdezés alatt?
  - Ezek a jelzők arra utalnak, hogy milyen eredmény ad vissza a lekérdezés. A skalár lekérdezés egy, a rekord több értéket ad vissza. A tábla lekérdezés eredményét több rekord, és általában több mező alkotja. A mező lekérdezés olyan tábla lekérdezés, amelyben csak egy mező szerepel.
4. Van-e alternatívája az allekérdezések használatának?
  - Egyes feladatok csak allekérdezéssel végezhetőek el, még mások az SQL más nyelvi elemeivel is megvalósíthatók. Utóbbi esetben az adatbázis-kezelője döntheti el, hogy melyik megoldást részesíti előnyben. Mindenki

azt a lehetőséget használja, amelyik a számára a legkényelmesebbnek és leglogikusabbnak tűnik.

5. Mit tudunk meg az alábbi lekérdezés végrehajtásával?

```
SELECT rDatum
FROM rendeles r_out natural join rendelestermek rt_out
  inner join
  (
    SELECT t_in.idTermek idTermek, SUM(rt_in.Darabszam) db
    FROM rendelestermek rt_in NATURAL JOIN termék t_in
    GROUP BY t_in.idTermek
    ORDER BY db DESC
    LIMIT 1
  ) toptermek
ON rt_out.idTermek=toptermek.idTermek
ORDER BY rDatum DESC
LIMIT 1;
```

- Az allekérdezés meghatározza annak a terméknek az azonosítóját, amelyből a legtöbbet adtak el. A főlekérdezés kiválasztja a rendeléseket és a rendelt tételeket. A fő és allekérdezés együtt alkot rekordforrást, ezért csak azokat a rendeléseket választjuk ki, amelyekhez a 'toptermék' kapcsolódik. A rendelési idő szerint csökkenő sorrendbe rendezett rekordok közül az első adjuk vissza. Röviden, az alábbi kérdésre válaszolunk:

Mikor rendeltek utoljára a legnépszerűbb termékből?





## 8. LECKE: REKORDOK BESZÚRÁSA AZ ADATBÁZIS TÁBLÁIBA

### 8.1 CÉLKITÚZÉSEK ÉS KOMPETENCIÁK

Eddigi munkánkat a DQL résznyelv választó és összesítő lekérdezéseinek segítségével végeztük. Előző fejezetünkkel az adatbázis kezelés talán legizgalmasabb részét, a tárolt adatok lekérdezésekkel történő kinyerését zártuk le. A most következő fejezet talán kevésbé alkalmas arra, hogy szárnyakat varázsoljon alkotó fantáziánkra, az azonban vitathatatlan, hogy a DML résznyelv most következő utasításaival megvalósítható feladatok mindennaposak, és megkevlhetetlenek az adatbázis-kezelésben.

Most következő leckénkben azt tanuljuk meg, hogyan lehet az adatbázis tábláiban új rekordokat elhelyezni. Bár ez a feladat nem tűnik túlságosan bonyolultnak, rövidesen látni fogjuk, mennyi apró részletre kell, és lehet ügyelnünk a megvalósításra alkalmas **INSERT** parancs használatakor.

A leckében megismerjük a parancs három általános formáját, amelyek a többféle felhasználás miatt első pillantásra bonyolultnak tűnhetnek.

A formai leírások megértésében sokat segít, ha rögzítjük, és betartjuk a rekordok beszúrásakor követendő általános szabályt:

Annyi, és sorrendben olyan típusú értéket szúrunk be, amennyit és amelyet a tábla definíciója megkíván.

Mivel egy rekord beszúrásakor nem mindig áll rendelkezésünkre az összes mező értéke, a fenti szabály betartása gyakran ütközik nehézségekbe. Rövidesen látni fogjuk azonban, hogy az **INSERT** megfelelő használatával elérhetjük, hogy a nem kötelező, az alapértelmezett értékkel rendelkező, vagy az automatikus sorszámozású, úgynevezett **AUTO\_INCREMENT** mezőket kihagyhassuk a parancsból.

Miután megértettük az **INSERT** egy rekord beillesztésre használható formáját, megbeszéljük, hogyan lehet a paranccsal egyetlen SQL-mondat felhasználásával több rekordot is beszúrni.

Ezt követően áttekintjük az **INSERT** parancsba allekérdezőként beágyazott választó lekérdezés eredményhalmazának felhasználását. Látni fogjuk, hogy ez a technika alkalmas egy tábla, vagy egy tábla részlet (mezők és rekordok) másik táblába másolására.

Az **INSERT** parancs használata után 180 fokos fordulatot téve, néhány gondolat erejéig visszatérünk a **SELECT** parancs használatához, és arról fogunk tanulni, hogyan lehet az eredményhalmazt fájlba menteni.

Ezzel egyrészt megtanuljuk az adatbázis adatainak egy egyszerű exportálási technikáját, másrészt alkalmat adunk arra, hogy megismerkedjünk **LOAD DATA INFILE** paranccsal, ami külső szövegfájlok tartalmának beolvasására és táblákba illesztésére alkalmas.

A lecke anyagának átolvasásakor az alábbi kérdések megválaszolására koncentráljon:

- Mik a rekordok beszúrásának legfontosabb szabályai?
- Milyen módszert biztosít az **INSERT** parancs a mezőértékek nevesített, és mezőnév nélküli beszúrására?
- Hogyan biztosítható, hogy egy mező az alapértelmezett értékét vegye fel?
- Hogyan érhető el, hogy az **AUTO\_INCREMENT** mezőkbe a következő automatikus érték kerüljön?
- Hogyan lehet egy **INSERT** paranccsal több rekordot beszúrni?
- Hogyan érhető el hogy az **INSERT**-tel beszúrt rekordokat egy választó lekérdezés eredményhalmaza adja?
- Miként lehet szövegfájlokban tárolt adatokat az adatbázis tábláiba írni?

## 8.2 AZ INSERT PARANCS ÁLTALÁNOS FORMÁJA

Az adatbázis tábláiba a DML résznyelvbe tartozó **INSERT** paranccsal szűrhatunk be új rekordot.

A parancs egyszerre egy vagy több rekord beillesztésére is alkalmas. A literálként megadott mezőértékek elhelyezkedhetnek közvetlenül az SQL mondatban, de a parancs alkalmas arra is, hogy a rekordokat és mezőértékeket egy választó lekérdezés biztosítsa.

Ennek megfelelően az **INSERT** általános leírása többféle lehet. Az egyes technikák alkalmazásakor azonban látni fogjuk, hogy az azokra alkalmas változat már sokkal barátságosabb és világosan megérthető. Elsőként lássuk az **INSERT** teljes specifikációját, majd pedig vizsgáljuk meg alkalmazási lehetőségeit!

```

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]
[IGNORE]
  [INTO] tábla [(mezőnév,...)]
  {VALUES | VALUE} ({kifejezés |
DEFAULT},...), (...), ...
  [ ON DUPLICATE KEY UPDATE
    mezőnév=kifejezés
    [,mezőnév=kifejezés] ... ]

```

vagy

```

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]
[IGNORE]
  [INTO] tábla
  SET mezőnév={expr | DEFAULT}, ...
  [ ON DUPLICATE KEY UPDATE
    mezőnév=kifejezés
    [, mezőnév=kifejezés] ... ]

```

vagy

```

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
  [INTO] tábla [(mezőnév,...)]
  SELECT ...
  [ ON DUPLICATE KEY UPDATE
    mezőnév=expr
    [, mezőnév=kifejezés] ... ]

```

## 8.3 REKORDOK BESZÚRÁSÁNAK SZABÁLYAI

Az **INSERT** parancs használata előtt fel kell elevenítenünk a táblák szerkezetével kapcsolatos tudásunkat.

A táblák, mint tudjuk mátrixok, mezők, és rekordok alkotta struktúrák. Míg a rekordok száma kötetlen, addig a mezők számát, sorrendjét, és különböző jellemzőit a tábla létrehozáskor deklaráljuk. Új rekord beszúrásakor, a mezők típusa, kötelező mivolta, esetleges alapértelmezett értéke, illetve automatikus számozása a legfontosabb mezőtulajdonság.

Amikor az **INSERT** parancssal új rekordot helyezünk el egy táblában, valóban a rekord mezőértékeit soroljuk fel literálok formájában.

Annyi mezőértéket kell megadni, amennyi mezőt a tábla deklarációja tartalmaz. A mezőértékeket a táblában meghatározott sorrendben kell feltüntetni.

A felsorolt mezőértékek típusának kompatibilisnek kell lenni a sorrendben megfelelő mezővel.

Ha a beszúrásakor nem áll rendelkezésre az összes mezőérték, akkor világosan jelezni kell, hogy a megadott literál melyik mezőhöz tartozik, és hogy mi történjen azokkal a mezőkkel, amelyek értékét nem adtuk meg.

### 8.3.1 Táblaterv megtekintése

Mivel a rekordok beszúrásakor fontos a tábla deklarációjának ismerete, szükségünk lehet a tábla szerkezetének megjelenítésére.

Erre a **SHOW COLUMNS FROM** *tábla* DCL-parancs a leginkább alkalmas. A hatására megjelenő sorok felsorolják a tábla mezőit, és azok minden fontos tulajdonságát: nevét, típusát, azt hogy lehet -e **NULL** értékű (azaz üres), mi az alapértelmezett értéke, hogy a mező kulcs mező-e, és hogy automatikus sor-számozású mezőről van-e szó.



A következő példa a **webbolt** adatbázis **vevo** táblájának szerkezetét jeleníti meg:



```
show columns from vevo;
```

Field	Type	Null	Key	Default	Extra
idVevo	int(10) unsigned	NO	PRI	NULL	auto_increment
vNev	varchar(50)	NO		NULL	
vTelefon	varchar(50)	NO		NULL	
vVaros	varchar(50)	NO		Eger	
vUtca	varchar(50)	NO		NULL	
vIRSZ	varchar(4)	NO		3300	
vMail	varchar(30)	YES		NULL	

38. ábra A vevo tábla szerkezete



Az ábrán jól láthatók a következők:



Az **int** típusú **idVevo** mező az elsődleges kulcs. Nem lehet **null** értékű (tehát kötelező), nincs alapértelmezett értéke, ugyanakkor automatikus számozású.



A **vNev**, **vTelefon**, **vVaros**, **vUtca**, **vIRSZ** mezők mind szövegek, és kötelezőek. A **vVaros** mező '**Eger**' a **vIRSZ** '3300' alapértelmezett értékkel rendelkezik.



A **vMail** mező használata nem kötelező, alapértelmezett értéke nincs megadva.

## 8.4 AZ INSERT OPCIONÁLIS ELEMEI

Az INSERT parancs után több opcionális, egyébként meglehetősen ritkán használt kulcsszó is elhelyezhető. Ezek az egymást kizáró LOW\_PRIORITY, DELAYED, és HIGH\_PRIORITY illetve az előző háromtól függetlenül alkalmazható IGNORE.

- LOW\_PRIORITY: akkor fejthet ki hatást, ha a táblát más is használja, miközben rekordokat szeretnénk benne elhelyezni. A LOW\_PRIORITY kulcsszó hatására DBMS mindaddig várakozik az INSERT végrehatásával, amíg a másik felhasználó olvasási művelete be nem fejeződik.
- DELAYED: Sok rekord egyidejű beszúrásakor használatos. A szerver már nem várja meg az összes rekord beszúrását, hanem vagy a következő parancsot kezdi feldolgozni, vagy visszaadja a vezérlést a felhasználónak.
- HIGH\_PRIORITY: Ellentétes LOW\_PRIORITY hatásával. Az INSERT párhuzamos hozzáférés mellett is azonnal elkezdődik.
- IGNORE: A kulcsszó a hibakezelésre van hatással. Ha alkalmazása mellett több rekordot szúrunk be egy táblába, és valamelyik rekord hozzáadása hibát eredményez, a végrehajtás nem szakad meg, csak a vétkes rekord beszúrása marad el.

## 8.5 REKORDOK BESZÚRÁSA MEZŐÉRTÉKEK FELSOROLÁSÁVAL

Az alábbiakban az INSERT használatának azokat a lehetőségeit tekintjük át, amelyek alkalmazásakor az SQL mondat tartalmazza az új rekord(ok) mezőértékeit.

### 8.5.1 Rekord beszúrása az összes mezőérték felsorolásával.

Az **INSERT** használatának legegyszerűbb módja, amikor az

```
INSERT táblanév VALUES (mezőérték, mezőérték ...)
```

formában adjuk meg a tábla egy rekordjének minden mezőértékét.

Ezt a módszert csak akkor használjuk, ha az összes mezőérték rendelkezésünkre áll.

**Mielőtt valamelyik mezőt kihagynánk (mert értékét nem akarjuk, vagy éppen nem tudjuk megadni) meg kell tudnunk, hogy a mező kötelező-e, illetve van-e alapértelmezett értéke!**

- Ha kötelező, és nincs alapértelmezett értéke sem, akkor nincs mit tennünk, meg **kell adni a mező értékét**.
- Ha kötelező, de a tábla deklarációjában meg van adva az alapértelmezett érték, akkor a mezőérték helyén feltüntetett **DEFAULT** kulcsszóval jelezhetjük, hogy ezt szeretnénk felhasználni.
- Ha mező nem kötelező, és van alapértelmezett értéke, akkor használhatjuk azt, ha nincs, akkor **NULL** értéket írhatunk a mezőbe.
- A kulcs mezők megadása mindig kötelező. Ha kulcs **AUTO\_INCREMENT**, akkor a **DEFAULT** érték feltüntetésével jelezzük, hogy a következő automatikus sorszámot kívánjuk használni.



A következő SQL mondat új rekordot szúr a **vevok** táblába:



**INSERT INTO vevok VALUES**



**(DEFAULT,**



**'Varga Jolán',**



**' (30) 451-2314 ',**



**DEFAULT,**



**'Kiskapu u, 14.',**



**DEFAULT,**



**NULL**



**);**

- Az első DEFAULT érték az IdVevo mező helyén áll. Mivel a mező AUTO\_INCREMENT, a DEFAULT kulcsszó a következő automatikus szám beszúrását jelenti.
- A második érték a név, amit a telefonszám követ.
- A vVaros alapértelmezett ('Eger') értéket kap.
- A vUtca értéke mag van adva,
- a vIRSZ ismét csak alapértelmezett ('3300').
- Az utolsó mező (vMail) nem kötelező, és nincs alapértelmezett értéke. Ha nem akarjuk kitölteni (a vevőnek nincs e-mail címe), NULL értéket kell kapnia.

Rekordok beszúrásakor szükségünk lehet, az utoljára fölvetett rekord azonosítójára. A MySQL rendelkezésre bocsátja a **LAST\_INSERT\_ID()** függvényt, ami a munkamenetünkben végrehajtott utolsó **INSERT** nyomán keletkezett automatikus azonosító értékét adja vissza.



**SELECT LAST\_INSERT\_ID();**

```
+-----+
| LAST_INSERT_ID() |
+-----+
|                31 |
+-----+
1 row in set (0.08 sec)
```

39. ábra LAST\_INSERT\_ID() függvény használata



A fenti példában minden mezőértéket megadtunk. Tegyük fel, hogy csak a kötelező és alapértelmezés nélküli mezők (**vNev**, **vTelefon**, **vUtca**) értékeit szeretnénk felsorolni.



**INSERT INTO vevo (vNev,vTelefon,vUtca)**



**VALUES (**



**'Zórád László',**



**'(60) 567-1412',**



**'Kicsi u, 2.'**



**);**



Figyeljük meg, hogy a **VALUES** után most csak a mezőnevek felsorolásának megfelelő mezőértékeket adtuk meg. A többi mező automatikusan **DEFAULT**, vagy **NULL** értéket kap. A következő ábrán a vevo tábla új rekordjai is látszanak.

idUvevo	vNev	vTelefon	vVaros	vUtca	vIRSZ	vMail
1	Andrássy Mária	<60> 832-4378	Budapest	Kassa utca , 94	1011	andrássy.maria@gabriel.hu
2	Dallos Rebeka	<30> 584-4836	Gyöngyös	Máza utca , 5	3200	dallos.rebeka@mail.hu
3	Gámb Huba	<60> 235-5859	Eger	Kolmann utca , 55	3300	galamb.huba@gabriel.hu
4	Károly Bertalan	<30> 891-6709	Eger	Cserezhnyés utca , 74	3300	károly.bertalan@post.hu
5	Keller Bert	<20> 942-4648	Miskolc	Eörsény utca , 33	3501	keller.berta@ansver.com
26	Andrássy Mária	<30> 832-4378	Budapest	Kassa utca , 94	1011	andrássy.maria@gabriel.hu
27	Andrássy Mária	<30> 832-4378	Budapest	Kassa utca , 94	1011	andrássy.maria@gabriel.hu
28	Andrássy Mária	<30> 832-4378	Budapest	Kassa utca , 94	1011	andrássy.maria@gabriel.hu
29	Andrássy Mária	<30> 832-4378	Budapest	Kassa utca , 94	1011	andrássy.maria@gabriel.hu
30	Andrássy Mária	<30> 832-4378	Budapest	Kassa utca , 94	1011	andrássy.maria@gabriel.hu
31	Andrássy Mária	<30> 832-4378	Budapest	Kassa utca , 94	1011	andrássy.maria@gabriel.hu
32	Zórád László	<60> 567-1412	Eger	Kicsi u, 2.	3300	NULL

40. ábra A vevo tábla a rekordok beszúrása után



### 8.5.2 Rekord beszúrása az mezőnevek és mezőérték felsorolásával.

Ha ezt a technikát alkalmazzuk, akkor a táblanevet követő kerek zárójelekben rendre fel kell sorolnunk a mezőneveket, majd a **VALUES** kulcsszót követő zárójelek között a mezőneveknek megfelelő sorrendben megadott mezőértékeket.

A mezőnevek sorrendje ilyenkor eltérhet a tábla mezőinek sorrendjétől, sőt, a nem kötelező, vagy alapértelmezett értékkel rendelkező mezőket el is hagyhatjuk. A kihagyott mezők automatikusan **DEFAULT** értéküket kapják.

Az egyetlen rekord mezőnevekkel együtt megadott mezőértékeinek beszúrásakor alternatív lehetőség a **SET** kulcsszó használata, ami után több *mező-név=mezőérték* értékadást sorolhatunk fel.



*Az előző INSERT így is helyes lett volna:*



**INSERT INTO vevo SET**



**vNev= 'Zórád László',**



**vTelefon= ' (60) 567-1412',**



**vUtca= 'Kicsi u, 2. '**



**;**

### 8.5.3 Több rekord beszúrása:

Az **INSERT** leírásából kiderül, hogy a **VALUES** kulcsszó után, egymástól vesszővel elválasztva, több kerek zárójelpár is elhelyezhető. Minden zárójelpár egy-egy rekord mezőértékeit tartalmazza. A mezőértékek sorrendje, meg kell feleljen az INSERT mező meghatározásainak, vagy tábla tervének.



*Az előző példákban beillesztett két rekordot egyszerre is beszúrhatuk volna.*



**INSERT INTO vevo (vNev,vTelefon,vUtca)**



**VALUES**



**( 'Zórád László', '(60) 567-1412', 'Kicsi u, 2.' ),**



**( 'Varga Jolán', '(30) 451-2314', 'Kiskapu u, 14.' )**



**;**

## 8.6 REKORDOK MEGADÁSA ALLEKÉRDEZÉSEKKEL

Az előző példákban a **VALUES** kulcsszót követő, kerek zárójelek között adtuk meg a beszúrandó rekordok mezőértékeit.

Ha ezek az az értékek az adatbázis más tábláiból alkotott választó lekérdezésekkel is kiválogathatók, az **INSERT** parancsban allekérdezéssel is megadhatjuk, a beszúrandó rekordokat.



Az INSERT parancsba ágyazott allekérdezést nem tesszük kerek zárójelek közé!

Az allekérdezés eredményhalmazát alkotó mezők sorrendjének és típusának meg kell felelnie az **INSERT**-ben megadott tábla, vagy a parancsot követő mezőnevek specifikációjának.



*Tegyük föl, hogy adatbázisunkban rendelkezésre áll az **egri** nevű tábla, és az a szándékunk, hogy ide másoljuk az **egri** lakhelyű vevők*

adatait. Mivel a tábla csak az egrieket tartalmazza, **vVaros**, és **vIRSZ** mezőkre nincsen szükség.



**SHOW COLUMNS FROM egri;**

Field	Type	Null	Key	Default	Extra
idUevo	int(10) unsigned	NO		0	
vNev	varchar(50)	NO		NULL	
vTelefon	varchar(50)	NO		NULL	
vUtca	varchar(50)	NO		NULL	
vMail	varchar(30)	YES		NULL	

41. ábra Az egri tábla szerkezete



A következő lekérdezés az **egri** táblába másolja az egri vevők adatait:



**INSERT egri**



**SELECT**



**null,**



**vNev,**



**vTelefon,**



**vUtca,**



**vMail**



**FROM** *vevo*



**WHERE** *vVaros='Eger'*



;

## 8.7 REKORDOK EREDMÉNYHALMAZÁNAK FÁJLBA MENTÉSE

A **SELECT** paranccsal kiválogatott eredményhalmazt eddig csak a kliens képernyőjén jelenítettük meg. A parancsot **INTO OUTFILE** záradéka azonban alkalmassá teszi arra, hogy az eredményhalmazt, annak mátrix szerkezetét megtartó szövegfájlba mentjük.

Az ilyen szövegfájlokban fontos kritérium, hogy jól elkülöníthetők legyenek a rekordok, és az azokon belül található mezőértékek. Ez elengedhetetlen ahhoz, hogy a fájlt később újra táblává lehessen alakítani.

Általában az állomány egyes sorai felelnek meg a rekordoknak. A rekordokban meghatározott módon elválasztva, azonos számú literálok írják le az egyes mezőértékeket.

Az alábbi formában megadott **SELECT** parancsban szabályozható a választó lekérdezés eredményének mentésére használt szövegfájl neve, karakterkészlete, a mezőket elválasztó jel, a mezőértékeket közrezáró karakterpár, az úgynevezett escape karakter (escape karakternek kell megelőzni a mezőértékek olyan karaktereit, amelyek azonosak a fájl tagolását biztosító valamelyik karakterrel), valamint a sorok (rekordok) kezdetét és végét jelző szövegek.

**A SELECT ... INTO OUTFILE kulcsszavakkal létrehozott szövegfájl nem a klienst futtató gép, hanem a szerver háttértárára, az aktuális adatbázis-könyvtárába kerül!**

Ha csak a fájlnevet adjuk meg, a DBMS az alapértelmezett kódolással menti az adatokat, úgy, hogy a rekordokat új sorokba írja, a mezőket vesszővel választja el, és a szövegmezőket nem zárja semmiféle jelpár közé.

```
SELECT mezőfelsorolás
FROM rekordforrás
INTO OUTFILE 'fájlnev'
  [CHARACTER SET karakterkészlet]
  [FIELDS | COLUMNS
    [TERMINATED BY 'szöveg']
    [[OPTIONALLY] ENCLOSED BY 'karakter']
    [ESCAPED BY 'karakter']
  ]
  [LINES
    [STARTING BY 'szöveg']
    [TERMINATED BY 'szöveg']
  ]
```



Az alábbi SQL mondat az **egriek.txt** nevű szövegfájlba menti a **vevo** tábla **vVaros="Eger"** feltételnek megfelelő rekordjait. A fájlba csak a **vNev**, **vTelefon**, **vUtca**, **vMail** mezők kerülnek:



**SELECT**



**vNev, vTelefon, vUtca, vMail**



**FROM vevo**



**WHERE vVaros='Eger'**



**INTO OUTFILE 'egriek.txt';**

## 8.8 SZÖVEGFÁJL TARTALMÁNAK BETÖLTÉSE AZ ADATBÁZISBA

A fenti módon készített szövegfájlok tartalma beolvasható, és az adatbázis valamely táblájába illeszthető a **LOAD DATA INFILE** paranccsal.

A parancs jó néhány lehetőséget biztosít, általános formája ennek megfelelően összetett.

```

LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE
'forrásfájl'
[REPLACE | IGNORE]
INTO TABLE tábla
[CHARACTER SET karakterkódolás]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'szöveg']
  [[OPTIONALLY] ENCLOSED BY 'karakter']
  [ESCAPED BY 'karakter']
]
[LINES
  [STARTING BY 'szöveg']
  [TERMINATED BY 'szöveg']
]
[IGNORE szám LINES]
[(mezőnév,...)]
[SET mezőnév = expr,...]

```

Használatakor a forrásfájl és a célzott tábla megjelölésén túl, világosan le kell írunk a betöltött fájl szerkezetét.



Tegyük fel, hogy adatbázisunk **egri** nevű táblája létezik.

Field	Type	Null	Key	Default	Extra
idUevo	int(10) unsigned	NO		0	
vNev	varchar(50)	NO		NULL	
vTelefon	varchar(50)	NO		NULL	
vUtca	varchar(50)	NO		NULL	
vMail	varchar(30)	YES		NULL	








42. ábra Az egri tábla



A következő SQL-mondat beolvassa az aktuális adatbázis könyvtárában lévő *egriek.txt* nevű állományt és tartalmát az **egri** táblába másolja.



**LOAD DATA INFILE 'egriek.txt'**

```
 INTO TABLE egri (  
  
 vNev,  
  
 vTelefon,  
  
 vUtca,  
  
 vMail  
  
 )  
  
 ;
```

## 8.9 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 8.9.1 Összefoglalás

Jelenlegi leckénkben az adatbázis-kezelés nélkülözhetetlen parancsa, a DML résznyelvbe tartozó INSERT használatát tekintettük át. A parancs új rekordo(ka)t szúr egy táblába, majd a rekordot feltölti a mondatban elhelyezett mezőértékekkel. Célját tekintve egyszerű parancs, de végrehajtásának, és ennek megfelelően SQL-mondatban történő megfogalmazásának több formája is van.

Használata előtt fontos megtanulni, hogy a rekordok beszúrásakor megadott mezőértékeknek számban, sorrendben és típusban is meg kell felelniük a bővített tábla mezőspecifikációjának.

Ezt vagy úgy érjük el, hogy az új rekord minden mezőértékét megadjuk, vagy úgy, hogy a parancsban világosan megfogalmazzuk, hogy a megadott mezőértékek melyik mezőhöz kapcsolódnak. Ennek érdekében a táblanév magadása után, zárójelek között felsorolhatjuk a megfelelő mezőneveket, vagy a **SET** kulcsszót követő **mezőnév=érték** értékadásokat használhatjuk.

A mezőértékek megadásakor – amennyiben a kérdéses mezők specifikációja ezt megengedi – lehetőségünk van az alapértelmezett (DEFAULT) illetve NULL értékek használatára is.

Az INSERT parancsban több rekord mezőértékeit is megadhatjuk. Erre VALUES kulcsszót követő kerek zárójelpárokban, vagy allekérdezés megfogalmazásával van lehetőségünk.

A rekordok beszúrásával foglalkozó leckénkben ismertük meg a LOAD DATA INFILE parancsot is. A LOAD DATA INFILE a DBMS hostjának valamely háttértárán található, megfelelő szerkezetű szövegállomány beolvasására és tartalmának táblába másolására alkalmas. Ilyen külső szövegfájlok készíthetők a SELECT INTO OUTFILE formátumú SQL-mondatokkal, de szinte bármilyen adatkezelő alkalmazással is. A LOAD DATA INTO így a külső adatforrások adatainak egyszerű importálására is használható.

## 8.9.2 Önellenoőrző kérdések

1. Mire kell odafigyelni az INSERT parancsban felsorolt mezőértékekkel kapcsolatban?
  - Arra, hogy számban, sorrendben, és típusban is megfelelő mezőértékeket soroljunk fel.
2. Mi a teendő azzal a mezővel kapcsolatban, ami kötelező, de van alapértelmezett értéke?
  - Ha nem adtuk meg a mezőneveket, akkor az ilyen mező értékét fel kell sorolni az INSERT parancs VALUES kulcsszava után. Ha az érték DEFAULT, akkor a DBMS az alapértelmezett értéket szőrja az új rekord mezőjébe. A mezőnevek felsorolása esetén csak akkor kell megadni a kérdéses mező értékét, ha a mező szerepel a felsorolásban.
3. Hogyan tudjuk egy meglévő tábla rekordjait vagy azok egy részét új táblába másolni?
  - Az INSERT parancsba ágyazott választó lekérdezéssel kiválasztjuk a megfelelő rekordokat és mezőket, amelyeket az INSERT beilleszt a célként megjelölt táblába.
4. Mire használható a SELECT parancs INTO OUTFILE záradéka?



- Ezzel a záradékkal irányíthatjuk a `SELECT`-tel létrehozott eredményhalmazt külső szövegállományba. A fájl a szerver host-on jön létre.
5. Hogyan olvasható be az adatbázis valamely táblájába, egy rekordokra és mezőkre tagolt szövegfájlok tartalma?
- Erre a feladatra a `LOAD DATA INFILE` parancs használható. A parancsot úgy kell paraméterezni, hogy az alkalmazkodjon a szövegfájl szerkezetéhez.



## 9. LECKE: REKORDOK MÓDOSÍTÁSA, ÉS ELTÁVOLÍTÁSA

### 9.1 CÉLKITÚZÉSEK ÉS KOMPETENCIÁK

Miután megtanultuk a táblákat rekordokkal feltöltő **INSERT** parancshoz kapcsolódó ismereteket, meg kell ismerkednünk a DML résznyelv további két elemével, az **UPDATE**, a **DELETE** parancsokkal, illetve DDL részét képező **TRUNCATE**-tel.

Ahogy korábban is tapasztaltuk, a kulcsszavak funkciója olyan röviden elmondható, hogy közben nem is gondolnánk, mennyi féle variációs lehetőség rejlik mögöttük.

A **DELETE** és **TRUNCATE** parancsok egyaránt rekordok törlésére alkalmasak, de még **TRUNCATE** barbár egyszerűséggel törli, azután újra létrehozza a táblát, a **DELETE** működése finoman szabályozható.

Az **UPDATE**, gyakran alkalmazott parancs. A tárolt adatok megváltoztatására, frissítésére használatos.

Most következő leckénk ezeket a parancsokat, azok nyelvtani leírását, használatuk jellemző módjait mutatja be, a **webbolt** adatbázis tábláin alkalmazott példákkal.

Mindhárom fenti parancs visszafordíthatatlan változásokat idézhet elő az adatbázisban, ezért fontos néhány szót szólnunk az elhibázott felhasználói lépések nyomán keletkező adatvesztésről, és a védekezés lehetséges módjairól. Tankönyvünk elsősorban az adatbázis-menedzser tevékenységeire és feladataira koncentrál, az adatbiztonsággal kapcsolatos feladatok megoldása pedig inkább az adminisztrátor tevékenységi körébe tartozik. Most mégis célszerű néhány szót ejteni azokról a lehetőségekről, amelyek viszonylag korlátozott jogok mellett is alkalmasak adataink mentésére.

A lecke figyelmes olvasása után Ön képes lesz biztonsági másolatokat készíteni adatbázisa tábláiról, biztonsággal el tudja végezni a rekordok változtatását, és tudni fogja, hogyan távolíthat el rekordokat a táblákból.

A leckében az alábbi problémákra, és kérdésekre keressen megoldásokat:

- Hogyan menthetjük adatainkat egy nagyobb horderejű adatváltoztatás előtt?

- Hogyan lehet megváltoztatni egy tábla egy, vagy több kiválasztott mezőjének értékét?
- Hogyan szűrhetjük a frissíteni kívánt rekordokat?
- Milyen lehetőségek vannak az **UPDATE** több felhasználós környezetben való használatára?
- Hogyan választhatjuk ki a törlendő rekordokat **DELETE** használatakor?
- Hogyan szabályozható a **DELETE**-tel törölni kívánt rekordok köre?
- Mi a **DELETE** és **TRUNCATE** parancsok közötti hasonlóság, és miben különbözik a két parancs?

## 9.2 ALAPFOKÚ ADATBIZTONSÁG

Az adatbázisokban tárolt adatok biztonságát veszélyeztető tényezők között előkelő helyet foglal el az emberi hiba.

Munkánk közben előfordulhat, hogy rekordjaink megváltoztatásakor, eltávolításakor nem megfelelő feltételt adunk meg, szélsőséges esetben egyáltalán nem szabályozzuk a műveletbe bevont rekordok körét. Az első esetben nem a megfelelő, a másodikban pedig a kiválasztott tábla minden rekordját módosítjuk, vagy töröljük. Bár ritkábban fordul elő, de hasonló, vagy talán még nagyobb katasztrófát idézhet elő a táblák meggondolatlan törlése.

Az ilyen felhasználó hibák elleni védekezés egyetlen lehetősége, az adatbázis rendszeres biztonsági mentése, amely meglehetősen összetett feladat révén, kiemelt jogosultságokat igényel, és az adatbázis-adminisztrátor tennivalói közé tartozik.

A DBMS-ek adminisztrátorai a megfelelő mentési beállítások mellett működő adatbázis-kezelő rendszerben képesek visszaállítani az adatbázis korábbi, konzisztens állapotát, azaz semmissé tudják tenni a véletlen felhasználói hibákat.

Mindamellett a rendszergazdák feladatköre meglehetősen összetett. Az adminisztrátorok általában stressztől terhelt, egyszerre ezer dologgal foglalkozó szakemberek.

A kedves olvasó talán el tudja képzelni azt, adatbázis adminisztrátor egyébként általában kifejezéselén arcáról elrugaszkodó, neandervölgyi ösztönytől táplált pillantást, amelyet a következő mondat vált ki:

– Ööööö, véletlenül töröltem az összes idei megrendelést...

Nos, az ilyen, és hasonló eseteknek elébe mehetünk, ha a DML-műveletek előtt biztonsági mentést, másolatot készítünk az inkriminált tábláról.

Az adatok szövegfájlba másolásának egyik technikájáról az előző leckében olvashattunk. Most a probléma különösebb körüljárása nélkül álljon itt a táblák adatbázison belüli másolásának néhány lehetősége. A következő SQL-mondatok DDL elemeket is tartalmaznak, tehát túlmutatnak a DQL, és DML résznyelveken.

#### Tábla másolatának létrehozása:

Amikor egy tábláról másolatot készítünk, előbb létre kell hozni az eredetinek megfelelő szerkezetű új táblát, majd belemásolni a menteni kívánt tábla rekordjait.

Erre a legegyszerűbb módszer a **CREATE** parancs alábbi használata:

```
CREATE TABLE másolt_tábla (SELECT * FROM  
forrás_tábla);
```

A megoldás alkalmas a rekordok helyes mentésére. A másolt tábla mezőinek típusai a forrás táblának megfelelőek lesznek. A kulcs, és index tulajdonságok ilyenkor nem kerülnek másolásra

Ha valamilyen oknál fogva csak a tábla rekordoktól mentes, strukturális másolatát szeretnénk elkészíteni, használjuk az következő parancsot:

```
CREATE TABLE másolt_tábla LIKE forrás_tábla;
```

Ha ezek után mégis be szeretnénk másolni a forrás rekordjait is, dolgozunk allekérdezést használó **INSERT**-tel:

```
INSERT másolt_tábla SELECT * FROM forrás_tábla;
```

A nem megfelelően végrehajtott DML-lekérdezések visszavonásának további lehetőségét biztosítja a tranzakciókezelés, amiről a következő leckében olvashatunk.

## 9.3 REKORDOK FRISSÍTÉSE

Az adatbázis kiválasztott rekordjaiban tárolt mezőértékek megváltoztatására az **UPDATE** parancsot használjuk.

A feladat végrehajtásakor ki kell választanunk a megfelelő táblát, és rekordokat. Ezt követően meg kell adnunk a frissítendő mezőket, valamint új értéküket.

Mivel az adatbázist egyszerre több felhasználó is használhatja, előfordulhat, hogy egy időben többen szeretnének módosítani egy rekordot. Az ilyen ütközések feloldása az DBMS beépített, és az adatbázis adminisztrátor által szabályozott funkciói közé tartozik. Bizonyos szinten azonban saját belátásunk szerint is segíthetjük az ütközések kezelését.

Az **UPDATE** szintaktikai leírásában láthatjuk majd, hogy a parancs minden feladatok megoldására kínál nyelvi eszközöket.

Az alábbiakban egy tábla, és az egyszerre több tábla rekordjainak módosításakor használható formákat látjuk:

#### Egy tábla módosítása:

```
UPDATE [LOW_PRIORITY] [IGNORE] tábla
    SET mező1 = kif1 | DEFAULT [, mező2 = kif2 |
    DEFAULT] ...
    [WHERE logikai_kifejezés ]
    [ORDER BY rendezés]
    [LIMIT sorok_száma ]
```

#### Több tábla módosítása:

```
UPDATE [LOW_PRIORITY] [IGNORE] rekordforrás
    SET mező1 = kif1 | DEFAULT [, mező2 = kif2 |
    DEFAULT] ...
    [WHERE logikai_kifejezés ]
```

### 9.3.1 Többfelhasználós környezet

Mindkét szintaxisban használhatók a **LOW\_PRIORITY** és **IGNORE** kulcsszavak.

A **LOW\_PRIORITY** akkor fejthet ki hatást, ha módosítással egy időben más felhasználók éppen kiolvassák az érintett rekordokat. A kulcsszó használata esetén az adatbázis-kezelő rendszer mindaddig várakozik az **UPDATE** végrehajtásával, amíg a másik felhasználó olvasási művelete be nem fejeződik. (Ez a funkció csak MyISAM, MERGE, és MEMORY típusú tábláknál működik.)

Az **IGNORE** kulcsszó az **UPDATE** végrehajtása közben bekövetkező hibák kezelését szabályozza. Alaphelyzetben a több rekordot is módosító parancs működése közben bekövetkező hiba esetén teljes visszavonás következik be. Az **IGNORE** használatakor csak a hibát okozó lépés nem kerül végrehajtásra.

Ha például úgy módosítunk táblát, hogy abban duplázódik egy kulcsmező értéke, a mondatban szereplő **IGNORE** hatására csak a hibát okozó mező frissítése marad el, a mondat többi része végrehajtódik.

### 9.3.2 Rekordforrás magadása

Az **UPDATE** a kiválasztott rekordokban megváltoztatja a megadott mezők értékeit. Helyes működéséhez pontosan meg kell jelölnünk a frissítendő rekordokat tartalmazó táblát, vagy táblákat.

A parancs azért rendelkezik két eltérő szintaxissal, mert használható egy bizonyos táblából származó rekordok, és több tábla kapcsolatával megadott rekordforrás rekordjainak frissítésére is.

Az első esetben az **UPDATE** [**LOW\_PRIORITY**] [**IGNORE**] szavakat követően egyetlen (a frissítendő) tábla neve, a második esetben több táblából álló rekordforrás szerepelhet. Utóbbit a **SELECT** parancs **FROM** záradékánál megtanult módszerekkel adhatjuk meg. Ebben esetben nem használhatjuk az **ORDER BY**, és a **LIMIT** záradékokat, amelyek **UPDATE** beli jelentéséről néhány bekezdéssel lejjebb rövidesen beszámolunk.

Amikor az **UPDATE** több tábla kapcsolatából alkotott rekordforrással dolgozik, előfordulhat, hogy a **SET** után megadott mező, a rekordforrás több táblájában is megtalálható. Ilyenkor nagyon fontos, hogy minősített hivatkozásokkal, szükség esetén álnevekkel tegyük egyértelművé a mezőhivatkozásokat.

### 9.3.3 Mezőértékek frissítése

Az **UPDATE** alapvető feladata, hogy megjelölt rekordforrás (egy vagy több tábla) rekordjaiban új értékeket adjon a kiválasztott mezőknek.

Erre az **INSERT** parancs esetében is alkalmazható **SET** kulcsszót a használhatjuk, amelyet, egy vagy több *mező=érték* formátumú értékadás követ. Az értékadásban megadott érték lehet literál, mezőhivatkozás, felhasználói változó, vagy a **DEFAULT** kulcsszó.

Miután a feltétlenül szükséges nyelvi elemeket áttekintettük lássunk egy konkrét példát a **webbolt** adatbázisban!



*Tegyük fel, hogy a szeretnénk nyilvántartani munkatársaink fizetését is. Kezdetben minden dolgozó 200 000 Ft keres majd, ezért a*

*munkatars* tábla eddig nem használt **mFizetes** mezőjét minden rekordban 200 000-re állítjuk.



```
UPDATE munkatars SET mFizetes=200 000;
```

### 9.3.4 Rekordok kiválasztása

A **SELECT** parancs esetén, a **WHERE** záradékkal szabályoztuk a kiválasztott rekordokat. A **WHERE**, az **UPDATE** parancsban is használható záradék, de itt a változtatásba bevont rekordok kiválogatására alkalmas. Az **UPDATE** csak azokban a rekordokban fogja átírni a mezők értékét, amelyekre a **WHERE** után megadott logikai kifejezés igaz értékű.



*Tegyük fel, hogy az LCD System nevű nagykereskedés jelzi, hogy a kapcsolattartó munkatársa neve megváltozott. Ezentúl nem Marton Ábel, hanem Késmárki Júlia munkakörébe tartozik a kiskereskedőkel való kapcsolattartás. A kapcsolattartó telefonszáma nem változik.*



*Ebben az esetben, a **nagyker** táblában az LCD System nevű cég rekordjában, az **nKapcsolat** mező értékét kell 'Késmárki Júlia'-ra változtatni:*



```
SELECT * FROM nagyker WHERE nCegNev='LCD System';
```



```
+-----+-----+-----+-----+
-----+
```



```
| idNagyker | nCegNev      | nKapcsolat | nTelefon
|
```



```
+-----+-----+-----+-----+
-----+
```





```
|          6 | LCD System | Marton Ábel | (30) 337-1588 |
```



```
+-----+-----+-----+-----+
-----+
```



```
UPDATE nagyker SET nKapcsolat='Késmárki Júlia'
WHERE nCegNev='LCD System';
```



```
SELECT * FROM nagyker WHERE nCegNev='LCD System';
```



```
+-----+-----+-----+-----+
-----+
```



```
| idNagyker | nCegNev      | nKapcsolat      | nTele-
fon           |
```



```
+-----+-----+-----+-----+
-----+
```



```
|          6 | LCD System | Késmárki Júlia | (30)
337-1588 |
```



```
+-----+-----+-----+-----+
-----+
```



Lássunk még egy példát!



*Király Oszkár munkatársunk által feldolgozott rendeléseket Létai Boglárka veszi át. Frissítsük a **rendeles** tábla megfelelő rekordjait.*



*Ebben az esetben az a feladat, hogy kiválasszuk a **rendeles** tábla azon rekordjait, amelyekkel eddig Király Oszkár dolgozott, majd a **rendeles** tábla **idRendeles** mezőjébe Létai Boglárka azonosítóját írjuk.*



*Az első példa kevésbé elegáns. Azt feltételezi, hogy előbb megnézzük, hogy Létai Boglárka azonosítója 3-as.*



**UPDATE rendeles NATURAL JOIN munkatars**



**SET rendeles.idMunkatars=3**



**WHERE munkatars.mNev='Király Oszkár';**



*A második megoldásban Létai Boglárka azonosítóját allekérdezéssel választjuk ki:*



**UPDATE rendeles NATURAL JOIN munkatars**



**SET rendeles.idMunkatars=(SELECT idMunkatars  
FROM munkatars WHERE mNev='Létai Boglárka')**



**WHERE munkatars.mNev='Király Oszkár';**

### 9.3.5 Sorrend és rekordszám

Az **UPDATE** parancs egytáblás rekordforrás esetén használható formájában alkalmazható az **ORDER BY** és a **LIMIT** záradék.

Az **ORDER BY** a **SELECT** esetén megismert módon képes szabályozni a kiválogatott rekordok, és ezen keresztül a módosítás sorrendjét. A **LIMIT** a módosításra kiválasztott rekordok számát határozza meg.

A két záradékot együtt alkalmazva elérhetjük, hogy egy valamilyen szempont szerint rendezett rekordforrásból, csak az első, megadott számú rekordot módosítsuk.



*Tegyük fel, hogy a legutolsó 10 rendelésben át szeretnénk írni a rendelést feldolgozó munkatárs azonosítóját. Ezeket a rendeléseket az 1-es azonosítóval (**idMunkatars**) rendelkező Rende Zsuzsánára bízuk. A feladat megoldását az alábbi sorok mutatják be:*



```
select idMunkatars, rDatum FROM rendeles  
ORDER BY rDatum DESC LIMIT 10;
```



```
+-----+-----+
```



```
| idMunkatars | rDatum |
```



```
+-----+-----+
```



```
|          4 | 2012-04-19 |
```



```
|          5 | 2012-04-08 |
```



```
|          9 | 2012-03-28 |
```





```
|          6 | 2012-03-16 |
```





```
|         10 | 2012-02-09 |
```


 | 4 | 2011-11-15 |




 | 10 | 2011-09-28 |

 | 2 | 2011-09-18 |


 | 8 | 2011-08-31 |


 | 3 | 2011-08-17 |


 +-----+-----+


 `UPDATE rendeles SET idMunkatars=1 ORDER BY  
rDatum DESC LIMIT 10;`
 `select idMunkatars, rDatum FROM rendeles  
ORDER BY rDatum DESC LIMIT 10;`
 +-----+-----+









 | idMunkatars | rDatum |

 +-----+-----+

 | 1 | 2012-04-19 |

 | 1 | 2012-04-08 |

 | 1 | 2012-03-28 |

		1		2012-03-16	
		1		2012-02-09	
		1		2011-11-15	
		1		2011-09-28	
		1		2011-09-18	
		1		2011-08-31	
		1		2011-08-17	
		+-----+-----+			

## 9.4 REKORDOK TÖRLÉSE

A **DELETE** parancs egy vagy több táblából álló rekordforrásokból képes rekordokat törölni. Amikor a rekordforrás csak egy táblából áll, az alábbi formátumot kell használni:

```
DELETE [LOW_PRIORITY] [IGNORE] FROM tábla_neve
      [WHERE logikai_kifejezés ]
      [ORDER BY rendezés]
      [LIMIT rekordszám ]
```

A **LOW\_PRIORITY** jelentése megegyezik az **UPDATE** parancsnál leírtakkal. MyISAM, MERGE, és MEMORY típusú táblákra alkalmazva az adatbázis-kezelő rendszer mindaddig késlelteti a törlést, amíg más felhasználók olvassák a tábla tartalmát.

Az **IGNORE** kulcsszó alkalmazása mellett a művelet akkor is folytatódik, ha valamelyik rekord törlése közben hiba keletkezik.

A **WHERE** záradékban adjuk meg a törlés feltételét.

Egy táblából álló rekordforrás esetén az UPDATE parancshoz hasonlóan használható az **ORDER BY** és **LIMIT** záradék.

Több táblából egyetlen **DELETE** paranccsal az alábbi formában távolíthatunk el rekordokat:

```
DELETE [LOW_PRIORITY] [IGNORE]
FROM tábla1 [, tábla2] ...
USING rekordforrás
[WHERE logikai_kifejezés]
```

Amikor több tábla kapcsolata alkotja a rekordforrást, a **DELETE** esetében sem használhatjuk az **ORDER BY** és a **LIMIT** záradékokat. Kötelező viszont a **USING** feltűntetése, amely záradékban a rekordforrás azon tábláit kell felsorolni, amelyekben el akarjuk végezni a törlést.

Első megközelítésben azt gondolhatjuk, hogy ha egyszer megadunk egy táblát a **DELETE** rekordforrásában, akkor ezt azért tesszük, mert a törlést erre a táblára is ki akarjuk terjeszteni. Ez azonban nem feltétlenül van így.



*Nézzük csak az alábbi példát!*



*Törölni akarjuk Andrásy Mária összes megrendelését.*



*Az alábbi példa többértelmű, ezért helytelen.*



*Nem lehet eldönteni, hogy csak a **vevo** táblából, a **rendeles** táblából, vagy mindkettőből törölni akarjuk a kiválasztott rekordokat.*



**DELETE**



**USING vevo NATURAL JOIN rendeles**



**WHERE**



***vevo.vNev='Andrássy Mária';***



*A helyes megoldás a következő:*



***DELETE***



***FROM      rendeles***



***USING    vevo NATURAL JOIN rendeles***



***WHERE***



***vevo.vNev='Andrássy Mária';***



*Abben a formában a rekordforrás megadásával válogattuk ki a tör-  
lendő rekordokat, de csak a **rendeles** táblából törltünk.*

## 9.5 TÁBLA KIÜRÍTÉSE

Az előző szakaszban megismert **DELETE** parancs az alábbi kontextusban minden rekordot töröl a kiválasztott táblából:

```
DELETE FROM tábla;
```

A **TRUNCATE** parancs látszólag azonos eredményt produkál, a művelete elvégzésében azonban van néhány különbség.

```
TRUNCATE [TABLE] tábla
```

- A **DELETE** számos paraméterrel rendelkezik, így szabályozható a törölni kívánt rekordok köre, száma, rekordforrása. A **TRUNCATE** egyetlen paramétere a tábla neve lehet.
- A **DELETE** sorról sorra törli a megadott táblák rekordjait. A **TRUNCATE** törli a táblát, majd az eredeti deklaráció szerint újra létrehozza.

- A **TRUNCATE** sokkal gyorsabban végzi el a feladatot, mint a **DELETE**.
- A **DELETE** tranzakció-biztos (lásd következő lecke), a **TRUNCATE** nem.
- A **DELETE** nem állítja vissza az **AUTO\_INCREMENT** mezők kezdőértékét, a **TRUNCATE** (működéséből adódóan) igen.
- A **DELETE** a DML résznyelvbe, a **TRUNCATE** inkább a DDL-be tartozik.

## 9.6 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 9.6.1 Összefoglalás

Mai leckénkben az adatok manipulálására használható parancsok közül az **UPDATE**, és **DELETE** használatával ismertük meg. Az **UPDATE** egy tábla, vagy több táblából alkotott rekordforrás forráshalmazának rekordjait, illetve azok mezőértékeit képes frissíteni. A paranccsal használható számos kulcsszó közül a **LOW\_PRIORITY** szabályozza a több felhasználós környezetben való működést, az **IGNORE** pedig a hibakezelésre van hatással. A frissítés a forráshalmazban, rekordról rekordra haladva történik úgy, hogy rendre végrehajtnak a **SET** kulcsszót követő értékadások. A **WHERE** záradékkal a művelet alá vont rekordok köre, az **ORDER BY** és **LIMIT** záradékokkal pedig sorrendjük illetve számuk szabályozható. Utóbbi két záradék csak akkor használható, amikor a rekordforrást egyetlen tábla alkotja.

A **DELETE** paranccsal rekordokat törölhetünk, a szintén egy vagy több táblából álló rekordforrásból. E parancs esetében az **UPDATE**-hez hasonló funkciókkal használhatók a **LOW\_PRIORITY** és az **IGNORE** kulcsszavak.

A parancs a **FROM** záradékban felsorolt táblák rekordjait törli. Több tábla esetén a **USING** záradékban kell megadni az azok közötti kapcsolatokat.

A műveletbe bevont rekordokat itt is a **WHERE** és a **LIMIT** záradékkal, a végrehajtás sorrendjét az **ORDER BY** záradékkal szabályozhatjuk. Az **ORDER BY** és a **LIMIT** csak egy táblából álló rekordforrások esetén használható.

Leckénk végén a **DELETE**, egy tábla összes rekordját törölő változatának alternatívájaként ismertük meg a **TRUNCATE** parancsot. Ez a parancs nem rekordokat töröl, hanem a teljes táblát, ezért inkább DDL résznyelvbe tartozik. Működésekor törli, majd ismét létrehozza a parancsban megadott táblát. Így az összes rekord törlése gyorsabb, azonban az **AUTO\_INCREMENT** a kezdeti értékre áll, és a parancs hatása nem visszaállítható.



### 9.6.2 Önellenőrző kérdések

1. Hogyan hozhatunk létre másolatot DML-művelet előtt egy tábláról?
  - `CREATE TABLE` másolat `SELECT * FROM` forrás
2. Mi a befolyásol az `UPDATE` parancs `IGNORE` kulcsszava?
  - Azt, hogy mi történjen a rekordok egymást követő frissítése közben bekövetkező hiba esetén. Ha nem használjuk a kulcsszót, hiba esetén a teljes művelet visszavonásra kerül és hibaüzenetet kapunk. Ha használjuk, akkor a hibát okozó frissítés nem valósul meg, de a frissítési művelet tovább folytatódik.
3. Hogyan korlátozhatjuk rekordszámra az `UPDATE` parancs működését?
  - A `LIMIT` záradékkal. A `LIMIT`-et gyakran az `ORDER BY` (sorrendet meghatározó) záradékkal együtt használjuk. Egyiket sem alkalmazhatjuk egyébként, több táblát érintő frissítésben.
4. Lehet-e a `DELETE` paranccsal csak kiválasztott mezőket törölni?
  - Nem. A `DELETE` csak teljes rekordokat töröl. A szabályozhatóság a rekordok tartalmára, számára, sorrendjére vonatkozhat.
5. Mi a különbség a `DELETE` és a `TRUNCATE` között:
  - A `DELETE` rekordokat távolít el egy táblából, a `TRUNCATE` az egész táblát törli, majd újra létrehozza.



# 10. LECKE: A MYSQL TOVÁBB- FEJLESZTETT LEHETŐSÉGEI

## 10.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

A MySQL eredeti tulajdonosa a MySQL AB nevű svéd szoftverfejlesztő cég, de a nyílt forráskódú adatbázis-kezelő rendszert előbb a Sun, majd 2010-ben az Oracle vásárolta fel. Utóbbi vásárlással kapcsolatban, sokakban fölvetődött az az aggály, hogy adatbázis-kezelő rendszerek terén hatalmas piaci részesedéssel rendelkező Oracle el akarja sorvasztani a saját termékével konkuráló MySQL-t.

Az aggodalom olyan erős volt, hogy az Európai Bizottság csak hosszas vizsgálódás, és Oracle 10 pontba foglalt szándéknyilatkozatának megjelenése után hagyta jóvá a tranzakciót. Az említett nyilatkozatban nem csupán a GPL-licencelést, hanem a megemelt költségekkel történő továbbfejlesztést, és folyamatos dokumentációt is megígérte a szoftveróriás. Az azóta elmúlt idő talán még nem elegendő ahhoz, messzemenő következtetéseket lehessen levonni a MySQL jövőjét illetően, az azonban bizonyos, hogy a szoftver jó úton halad afele, hogy az Oracle deklarált célját beváltva az MS-SQL szerverek méltó ellenfelévé váljon.

Bár e tekintetben a sebesség, a replikációs képességek, és a fürtözhetőség a legfontosabb szempontok, a modern adatbázis-kezelő rendszerekkel szemben megnövekvő igényeknek megfelelően, a MySQL-ben elérhetők, és egyre professzionálisabb szintre fejlődtek az olyan képességek, mint a tárolt eljárások és függvények, a tranzakció kezelés, vagy a triggerek.

Most következő leckénkben ezekről a lehetőségekről ejtünk szót. A lecke megtanulása után Ön a tranzakciók használatával biztosítani tudja majd adatkezelő műveleteinek visszaállíthatóságát, az adatbázis konzisztenciáját.

Képes lesz kisebb, egyébként az adatbázis alkalmazás feladatai közé tartozó műveleteket, tárolt eljárások formájában létrehozni és közvetlenül az adatbázisban elhelyezni. Ismerni fogja a saját munkáját megkönnyítő, a MySQL beépített függvényeit kiegészítő, felhasználói funkciók létrehozásának módját. Képes lesz az adatbázisban bekövetkező eseményekre, törlési, bővítési, változtatási műveletekre reagáló utasítások létrehozására.

## 10.2 TRANZAKCIÓK KEZELÉSE

Az adatbázis-kezelés során előfordulhatnak olyan adatkezelő műveletek, amelyek veszélyeztetik az adatbázis konzisztenciáját, az adatok hiba- és ellent-

mondásmentes tárolását. Az ilyen műveletek általában több lépésből, részműveletekből állnak. A részműveletek alatt az adatbázis inkonzisztenssé válhat, de a teljes művelet sor befejezésekor vissza kell állnia a konzisztens állapotnak. Az ilyen műveletek sikeres lebonyolítása hálózati környezetben bizonytalanná válhat.

Előfordulhat, hogy valamilyen hálózati, de egyébként bármilyen hiba miatt az adatbázis kezelője nem tudja végrehajtani az összes részműveletet, így az adatbázis inkonzisztens állapotban marad.

Az ilyen problémák kezelésére használhatók a tranzakciók.

A tranzakció olyan több lépésből álló adatbázis-kezelési műveletsorozat, amelyet a DBMS egy atomi egységben hajt végre. Ha a tranzakció minden lépése sikeres, az adatbázis új, konzisztens állapotba került. Ha bármely lépés hibát okoz, vagy az adatbázis kezelője úgy dönt, hogy mégsem akarja végrehajtani a műveleteket, akkor a teljes lépéssorozat visszavonható, és az adatbázis a korábbi konzisztens állapota áll vissza.



*Tegyük fel, hogy a **webbolt** adatbázisban új rendelést veszünk fel. A **rendeles** táblában rögzített rendeléshez a **rendeles-termek** táblában elhelyezett új rekorddal, hozzárendelünk egy terméket, és megadjuk annak darabszámát. Ezt követően a **termek** táblában csökkentenünk kell a megadott termék készletét a megrendelt darabszámmal.*



*Ha ebben a pillanatban valamilyen hiba következik be, a készlet-csökkentés elmaradhat, és az adatbázis inkonzisztens marad.*



*Hasonlóan kritikus helyzet állhat elő, ha el tudjuk ugyan végezni a készlet csökkentését, de 0-nál kisebb értéket kapunk. Ez ugyanis azt jelenti, hogy a termékből nem állt rendelkezésre a megrendelt mennyiség.*



*Ha ilyen szituációba kerülünk, vissza kell állítanunk a korábbi készletet, a rendelésből törölnünk kell a rendelt terméket, sőt lehet, hogy a rendelést is el kell távolítanunk. Az ilyen műveletek egy tranzakcióban való kezelése megoldja a problémát.*

### 10.2.1 A tranzakciók a MySQL-ben:

A MySQL az InnoDB típusú táblákban biztosítja a tranzakció kezelés lehetőségét. Fontos tudnunk, hogy csak a DML résznyelvbe tartozó parancsok tranzakcióbiztosak. Ez azt jelenti, hogy az egy tranzakcióban lévő parancsok közül csak a DML-be tartozók hatása tehető semmissé, vonható vissza.



Ezért mondtuk az előző lecke végén, hogy a DELETE visszaállítható a TRUNCATE nem.

#### Tranzakció indítása

A MySQL-ben a **BEGIN**, vagy **START TRANSACTION** parancsokat követő SQL mondatok egy tranzakcióba fognak tartozni. A tranzakció a **COMMIT** illetve a **ROLLBACK** parancsok kiadásával, vagy hibával zárulhat.

Hiba, vagy a **ROLLBACK** esetén a tranzakcióba tartozó utasítások semmissé válnak, visszakapjuk a tranzakciót megelőző állapotot. A **COMMIT** parancs kiadására a DMBS lezárja a tranzakciót, és rögzíti az új konzisztens állapotot. A **COMMIT** tehát a jóváhagyás, a **ROLLBACK** a visszavonás parancsa.



*Most vizsgáljunk meg egy egyszerű példát, ami demonstrálja a tranzakció kezelés működését.*



*Andrássy Mária nevű vevőnk új rendelést ad le, amelyben 3 db HP LaserJet 1015 típusú lézernyomtatót szeretne vásárolni.*



*A rendelést Tuba Zsuzsanna munkatárs fogja kezelni.*



*Elsőként a **rendeles** táblában kell új rekordot elhelyeznie, úgy hogy az **idMunkatars** mezőben saját, az **idVevo** mezőben pedig a vevő azonosítóját tárolja. Ezt követően a **rendelestermek** táblába fölvetett új rekordban, meg kell adnia a rendelés és a nyomtató azonosítóját, valamit 3-as darabszámot. Ezután már csak a termék készleten lévő számát kell csökkentenie.*



*Tegyük fel, hogy előrelátóan tranzakcióba foglalta a műveleteket!*



*A munkatárs az alábbi lépésekben végezte el a feladatot.*



*Rekord beszúrása a **rendeles** táblába:*



```
INSERT rendeles (idMunkatars,idVevo) VALUES (
```



```
(SELECT idMunkatars FROM munkatars
```



```
WHERE mNev='Tuba Zsuzsanna'),
```



```
(SELECT idVevo FROM vevo
```



```
WHERE vNev='Andrássy Mária')
```



```
);
```



*Query OK, 1 row affected (0.00 sec)*



*Rekord beszúrása a **rendelestermek** táblába:*



```
INSERT rendelestermek (idRende-  
les,idTermek,DarabSzam) VALUES (
```



```
LAST_INSERT_ID(),
```



```
(SELECT idTermek FROM termék
```



```
WHERE tMegnevezes LIKE 'HP LaserJet 1015%'
```



),



3



);

*Query OK, 1 row affected (0.00 sec)**Nyomtató rekordjának módosítása a termék táblában?**UPDATE termék SET tKeszlet=tKeszlet-3**WHERE tMegnevezes LIKE 'HP LaserJet 1015%';**Ezen a ponton hiba keletkezett, és az alábbi hibaüzenet jelent meg:**ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(`webbolt2`.`termek`.`tKeszlet` - 3)'**Ha megvizsgáljuk a **termek** táblát, hamar kiderül az ok. A **tKeszlet** mezőben 2-es érték van.**SELECT tMegnevezes, tKeszlet FROM termék**WHERE tMegnevezes LIKE 'HP LaserJet 1015%';*

+-----+-----+



| tMegnevezes | tKeszlet |



```
+-----+-----+
```



```
| HP LaserJet 1015 Kifutó! | 2 |
```



```
+-----+-----+
```



Ebből akartunk kivonni 3-at, az eredmény pedig negatív lett. A **tKeszlet** mező azonban **INT(10) UNSIGNED** típusú, azaz értéke csak pozitív lehet. Az utolsó művelet sikertelen. Ez azt is jelenti, hogy **rendeles**termek és a **rendeles** táblák utoljára föl-vett rekordjait is törölni kell, tehát mindent „vissza kell csinálni”! Ha a Tuba Zsuzsanna nem használt volna tranzakciót, most meg kellene keresnie a **rendeles**termek és a **rendeles** tábla utolsó rekordját, majd törölnie kellene azokat. A helyzetet az is nehezíti, hogy közben más munkatársak is rögzíthettek rendeléseket. Korántsem biztos, hogy az Andrassy féle rendelés rekordjai az utolsók!



Mivel azonban az egész művelet sor egy tranzakció részeként került végrehajtásra, most elég kiadni a **rollback** parancsot, és minden visszaáll a tranzakció előtt állapotba! Természetesen a mások által közben rögzített rendelések megmaradnak.

## 10.3 NÉZETEK

Bonyolultabb, és gyakran használt választó lekérdezések, gyakori futtatása esetén sokat könnyíthetünk saját munkánkon a nézetek segítségével.

A nézetek valójában névvel ellátott és tárolt DQL-parancsok, választó lekérdezések, amelyek eredményhalmazára táblaként hivatkozhatunk.

A nézeteket használatuk előtt deklarálni kell. A deklaráció során megadjuk a nézet nevét, és azt a DQL-mondatot, amely a nézethez tartozó lekérdezést végzi.

Miután egy nézet elkészült, az arra jogosult felhasználók úgy hivatkozhatnak rá, mintha tábla lenne. Fontos azonban tudni, hogy a nézetek rekordjai nem tárolódnak külön az adatbázisban. Ezek, a nézetbe foglalt lekérdezés ered-



ményhalmazát képezik, amit a DBMS, a nézet minden egyes használatakor újra előállít.



*Az alábbi példa olyan nézet készítését és használatát mutatja be, amely Király Oszkár nevű dolgozónk számára teszi lehetővé saját megrendeléseinek gyors áttekintését.*



***CREATE VIEW kiraly AS***



***SELECT \* FROM***



***rendeles***



***NATURAL JOIN rendelestermek***



***NATURAL JOIN termek***



***NATURAL JOIN categoria***



***WHERE rendeles.idMunkatars=***



***(***



***SELECT idMunkatars***



***FROM munkatars***



***WHERE mNev LIKE 'Király Oszkár'***



)



*ORDER BY rendeles.rDatum, termék.tMegnevezes*



;

A fenti nézet elkészítése után a nézet táblaként használható:



*SELECT \* FROM kiraly;*

## 10.4 TÁROLT ELJÁRÁSOK

Minden adatbázisban akadnak gyakori, de bonyolult módon összeállítható, vagy csak több SQL-mondattal elvégezhető feladatok.

A fejlett adatbázis-kezelő rendszerek lehetővé teszik egy, vagy több SQL-mondat nevesített, és az adatbázisban tárolódó eljárásba foglalását. Az ilyen, úgynevezett tárolt eljárások az eljárás nevével bármikor elindíthatók, és az eljárásban lévő utasítások végrehajthatók.



*Tegyük fel például, hogy gyakran akarjuk megtekinteni egy kategória termékeit! Tárolt eljárás nélkül, feladat megoldására az alábbi SQL-mondatot használjuk:*



*SELECT \*.termek FROM termék NATURAL JOIN categoria WHERE categoria.kNev='processzor';*



*A 'processzor' literár helyére mindig a megfelelő kategória nevét írjuk.*



*A fenti SQL mondat használata helyett létrehozhatnánk a **categoria\_lista** nevű eljárást, amely paraméterként átveszi egy kategória nevét, majd listázza a kategória termékeit. Ebben az esetben a fenti művelet ilyen egyszerűen is elvégezhető lenne:*



```
CALL kategoria_lista('processzor');
```

### 10.4.1 Tárolt eljárások létrehozása

A tárolt eljárások utasításait az első használat előtt rögzíteni, **deklarálni** kell. A deklarációban adjuk meg a tárolt eljárás **nevét**, **paramétereit**, és az eljáráshoz tartozó **utasításokat**. A deklaráció egyetlen SQL-mondattal történik, amit az éppen beállított **határoló karakter** zár.

A MySQL alapértelmezett határoló karaktere a **pontosvessző (;)**, de ezt a **DELIMITER** parancs használatával bármilyen szövegre lecserélhetjük.

Az **eljáráson belül**, a sorok végét – a beállított határoló karaktertől függetlenül – **mindig pontosvessző (;)** jelzi. Éppen ezért eljárás deklarációja alatt a **DELIMITER** értéke nem maradhat pontosvessző! A deklaráció előtt általában megváltoztatjuk, az eljárás végén pedig visszaállítjuk a **DELIMITER**-t.

```
DELIMITER //
    eljárás_deklarációja
DELIMITER ;
```

Ha a pontosvessző maradna a **DELIMITER**, akkor az eljárás első sorvégén befejeződne.

Az eljárás deklarációjának formális leírása a következő:

```
CREATE PROCEDURE eljárásnév([formális_paraméterlista])
BEGIN
    [utasítások]
END delimiter
```

### 10.4.2 Tárolt eljárás paramétereit

Az eljárások híváskor paramétereket vehetnek át. A paraméterek az eljárás deklarációjakor megadott formális paraméterlista változóiba kerülnek. A formális paraméterlista nem más, mint a paraméterváltozók, használati módjaik és adattípusaik felsorolása. Az eljárás híváskor a formális paraméterlistának megfelelő aktuális paramétereket kell átadni.

Az eljárás formális paraméterlistáját az alábbi formátumban határozhatjuk meg:

```
mód paraméternév típus(méret)
```

```
használati mód: IN|OUT|INOUT  
paraméternév: paraméterváltozó neve,  
típus: a paraméter típusa (MySQL mező típus)
```

Az IN módú paraméterekkel adatok vehetők át. Az OUT paraméterek adatok visszaadására használhatók. Helyükön az eljárás híváskor felhasználói változót kell megadni. Az eljárás működése közben megváltoztathatja az ilyen paraméter értékét, a változás pedig a hívás helyén megadott változóban is érvényesül. Az **INOUT** paraméterek helyén szintén felhasználói változónak kell szerepelnie, de az ilyen paraméterek adatok átvételére és visszaadására is alkalmasak.

Az eljárásokban lokális változókat használhatunk az adatok átmeneti tárolására. Ezeket természetesen a meghatározott formula szerint deklarálni kell az eljárás utasításai között.

```
DECLARE változónév [, változónév] típus [DEFAULT  
érték];
```

A deklarált helyi változóknak az alábbi formában adhatunk értéket:

```
SET változónév=érték;
```

### 10.4.3 Vezérlési szerkezetek

A tárolt eljárások érdekessége, hogy nem csak SQL-mondatokat, de szükség esetén ciklus- és elágazásszervező vezérlési szerkezeteket is tartalmazhatnak. Az alábbiakban ezek formai leírását tekintheti át az olvasó. A szintaxisok magukért beszélnek, részletes tárgyalásukra nem térünk ki.

## Elágazások

```
IF logikai_kif THEN parancsok
[ELSEIF logikai_kif THEN parancsok]
[ELSE parancsok]
END IF;

CASE
WHEN kifejezés THEN parancsok
...
WHEN kifejezés THEN parancsok
ELSE parancsok
END CASE;
```

## 10.4.4 Ciklusok

```
WHILE logikai_kif DO
parancsok;
END WHILE ;

REPEAT
parancsok;
UNTIL logikai_kif
END REPEAT;
```

## 10.4.5 Eljárások hívása

A létező eljárások a CALL paranccsal, a formális paraméterlistának megfelelő aktuális paraméterekkel hívhatók meg:

```
CALL eljárásnév([aktuális_paraméterlista])
```



*Ezek után lássuk, hogyan hozhatnánk létre az előbbi feladatot megoldó, `kategoria_lista` eljárást!*



**DELIMITER //**



**CREATE PROCEDURE `kategoria_lista`(IN `kat`  
VARCHAR(30))**

**BEGIN****SELECT \*****FROM termek NATURAL JOIN categoria****WHERE categoria.kNev=kat;****END //****DELIMITER ;****CALL categoria\_lista('nyomtató');**

idKategoria	idTermek	tMegnevezes	Nagyker	kNev
7	222	Samsung ML-2551N lézer, Dupl+Hálóz., PP+U	2	Nyomtató
7	223	Samsung SCX-4100 nyomtató-másoló-scanner	2	Nyomtató
7	224	Samsung SCX-4216F nyomtató-másoló-scanner	2	Nyomtató
7	232	HP LaserJet 1015 Kifutó!	2	Nyomtató

4 rows in set (0.00 sec)

43. ábra *categoria\_lista()* eljárás futtatása

## 10.5 TÁROLT FÜGGVÉNYEK

A tárolt függvények az eljárásokhoz hasonlóak, de hívásuk után eredményt, visszatérési értéket adnak vissza. Alkalmazásukkal saját, az aktuális adatbázis kezelésére alkalmas függvényeket építhetünk magába az adatbázisba.



*Tegyük fel, hogy rendszeresen szükségünk van különböző termék-kategóriák átlagos eladási árára. Az átlagárát viszonylag egyszerűen meghatározhatjuk, de a 'nyomtató' szó helyére az éppen szükséges kategória nevét írva mindannyiszor be kell gépelnünk az alábbi SQL-mondatokat:*



```
SET @kat='nyomtató'
```



```
SELECT AVG(tEladAr) FROM
```



```
termek NATURAL JOIN categoria
```



```
WHERE kNev=@kat GROUP BY kNev
```

Ha létezne a fenti műveleteket elvégző, és a számítás eredményét visszaadó függvény, az átlag kiírása ilyen egyszerű lenne:

```
SELECT katatlag('nyomtató');
```

A függvények deklarálása az eljárásoknak megfelelően történik két apró, de fontos kitéttel. Meg kell adni a függvény **visszatérési értékének**, a visszaadott eredménynek a **típusát**, és a függvényben valahol szerepeltetni kell **RETURN** parancsot, amellyel visszaadható az **eredmény**.

### 10.5.1 Függvény deklarálása

```
CREATE FUNCTION függvéynév(form_param_list) RETURNS  
eredmény  
BEGIN  
    [utasítások]  
    RETURN eredmény  
END delimiter
```



Egy termékkategória átlagos árát kiszámoló tárolt függvény a következőképpen készíthető el:



```
DELIMITER //
```



```
CREATE FUNCTION katatlag(kat VARCHAR(20))
```



*RETURNS FLOAT(10,2)*



*BEGIN*



*DECLARE atlag FLOAT(10,2);*



*SET atlag=(SELECT AVG(tEladAr)*



*FROM termék NATURAL JOIN  
kategoria*



*WHERE kNev=kat GROUP BY  
kNev*



*);*



*RETURN atlag;*



*END //*



*DELIMITER ;*

## 10.6 TRIGGEREK

A tárolt eljárásokat és függvényeket a felhasználó hívhatja meg, indíthatja el. Szükségünk lehet olyan, akár több műveletből álló **SQL-mondat** sorozatokra is, amelyek az adatbázisban bekövetkező valamilyen **változás hatására automatikusan elindulnak**. Ezt a lehetőséget biztosítják a triggererek.

A triggererek olyan, a tárolt eljárásokhoz némileg hasonló funkciót biztosító utasítássorozatok, amelyek az adatbázis valamelyik tábláján végrehajtott **törlés, módosítás, vagy beszúrás esetén automatikusan** elindulnak.





*Triggerre lehet szükségünk például, ha biztosítani szeretnénk, hogy egy rendelési tétel rögzítésekor automatikusan csökkenjen a megfelelő termék készleten lévő mennyisége.*

Az eljárásokhoz hasonlóan a triggereket is deklarálni kell. Ehhez alábbi formula használható:

#### **CREATE**

```
TRIGGER trigger_neve indulási_pont trigger_esemény
ON tábla_név FOR EACH ROW utasítások
```

A formai leírás elemei igényelnek némi magyarázatot.

- **trigger\_neve**: minden triggernek egyedi névvel kell rendelkeznie.
- **trigger\_esemény**: **INSERT** | **UPDATE** | **DELETE**. Annak a módosító műveletnek a neve, amely a trigger működését kiváltja.
- **indulási\_pont**: a **BEFORE** | **AFTER** értékekkel azt adjuk meg, hogy az utasítás végrehajtása előtt, vagy az után aktiválódjon a trigger.
- **tábla\_név**: a trigger által „figyelt” tábla neve. A trigger csak az ebben a táblában bekövetkező eseményekre reagál.
- **utasítások**: azok az SQL-mondatok, amelyeket végre kell hajtani, amikor a trigger aktiválódik.



A triggerekben nem használhatjuk a tranzakciókezelő **BEGIN**, **COMMIT**, **ROLLBACK** parancsolat. Ugyanakkor minden trigger automatikusan önálló tranzakciót alkot, ezért ha a trigger parancsai közül valamelyik hibával végződik, az összes többi utasítás hatása semmissé válik.



*Ezek után lássuk a fent felvetett probléma megoldását!*



**DELIMITER //**



**CREATE TRIGGER** termekcsokkenes



**BEFORE INSERT ON** rendelestermek



***FOR EACH ROW***



***BEGIN***



***DECLARE aktKeszlet INT(10) UNSIGNED;***



***#UNSIGNED, nem lehet negatív***



***DECLARE ujKeszlet INT(10) UNSIGNED;***



***SET aktKeszlet=(***



***SELECT tKeszlet***



***FROM termék***



***WHERE idTermek=NEW.idTermek***



***);***



***#Ha < 0, akkor hiba => ROLLBACK***



***SET ujKeszlet=aktKeszlet-NEW.DarabSzam;***



***UPDATE termék***



***SET tKeszlet=ujKeszlet***



```
WHERE idTermek=NEW.idTermek;
```



```
END //
```



```
DELIMITER ;
```



A fenti trigger létrehozása után valahányszor fölveszünk egy rendelési tételt **rendelestermek** táblába, a **termek** tábla megfelelő termékének készlete automatikusan csökken a darabszám értékével. Ha az új készlet negatív lenne, a teljes utasítássorozat semmisé válik.

## 10.7 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 10.7.1 Összefoglalás

Mai leckénkben a MySQL DBMS néhány, adatkezelési műveletek automatizálására, és az adatok konzisztens állapotának megőrzésére használható funkcióját tekintetük át.

Megismerkedtünk a több lépésből álló, összetett adatkezelő műveleteket egy atomi egységként kezelő tranzakciókkal. Megtanultuk, hogy a tranzakciók a **begin**, vagy **start transaction** paranccsal indíthatók, illetve a **commit** és a **rollback** parancsokkal fejezhetők be. A **commit** parancsot a tranzakció sikeres lezárására, és a változtatások rögzítésére, a **rollback** parancsot pedig a tranzakció előtti konzisztens állapot visszaállítására használjuk.

Megismerkedtünk a gyakran futtatott DQL-lekérdezések egyszerű használatára való, virtuális táblaként alkalmazható nézetekkel. Megtanultuk, hogy a nézet létrehozására alkalmas **CREATE VIEW** parancs után meg kell adnunk a nézet **nevét** valamit, azt a DQL-mondatot, amely a nézet felhasználásakor kerül végrehajtásra. Az elkészült nézetekre úgy hivatkozhatunk, mintha azok táblák lennének, holott a nézet eredményhalmaza nem tárolódik külön az adatbázisban.

Leckénk második felében a tárolt eljárásokkal és függvényekkel foglalkoztunk. A tárolt eljárások és függvények lehetővé teszik, hogy adatkezelő programkódokat építsünk be az adatbázisba. Ezzel elérhetjük, hogy az adatbázis

alkalmazás egyes feladatait, közvetlenül az adatbázis-kezelő rendszer végezze el.

Mai leckénket az adatbázisban bekövetkező eseményekre reagálni képes triggerek használatával fejeztük be.

### 10.7.2 Önellenőrző kérdések

1. Milyen utasításokkal kezelhetjük a tranzakciókat?
  - A tranzakciót **BEGIN** vagy a **START TRANSACTION** utasítással indíthatjuk. Sikeres végrehajtás és rögzítés esetén a **COMMIT**, visszaállítás esetén a **ROLLBACK** paranccsal fejezhetjük be.
2. Mire használhatók a nézetek?
  - Gyakran futtatott választó lekérdezés virtuális táblaként történő használatára.
3. Mi a különbség tárolt eljárás és függvény között?
  - A tárolt eljárás futásakor nem keletkezik visszatérési érték, a függvény esetében igen.
4. Mit tud elmondani az itt látható SQL mondatról?

```
CREATE FUNCTION atlagar()  
RETURNS FLOAT(10,2)  
BEGIN  
    RETURN (SELECT AVG(tEladAr)FROM termek);  
END
```

  - Az utasítás az **atlagar()** nevű beépített függvényt hozza létre. A függvény nem vár paramétert, visszatérési értéke **FLOAT** típusú. Eredményként az adatbázis termék táblájában található **tEladAr** mező átlagát adja vissza.
5. Mi a különbség tárolt eljárások és a triggerek között?
  - A tárolt eljárásokat a felhasználó indíthatja a **CALL** paranccsal. A triggerek indítását az adatbázis „figyelt” táblájában bekövetkező, adatkezelő műveletek indítják el.

# 11. LECKE: MYSQL DBMS ÉS ADATBÁZIS ALKALMAZÁSOK KOMMUNIKÁCIÓJA

## 11.1 CÉLKITŰZÉSEK ÉS KOMPETENCIÁK

A korábbi leckékben már foglalkoztunk azzal, hogy hogyan lehet a MySQL-adatbázisok adatait külső alkalmazásokkal feldolgozható formátumban, szövegállományokba exportálni. Megbeszéltük azt is, miként tudunk a mezőket és rekordokat világosan elkülönítő szövegállományokból, adatokat betölteni az adatbázisokba.

Most azt fogjuk megvizsgálni, hogy milyen lehetőségek vannak a kliensoldali alkalmazások és a MySQL-szerver kapcsolatának biztosítására, a szerver és kliens közötti szinkron adatcserére. A leckében megismerjük a MySQL-szerverek és kliensek közötti kommunikációhoz kapcsolódó fogalmakat. Megbeszéljük, mire való a MySQL-protokoll, és hogy milyen szerep jut implementációiban, a mysql-lib könyvtárnak, illetve a natív drivereknek. Kitérünk a connectorok, és az API-k közötti különbségekre majd egy-egy példán keresztül megvizsgáljuk használatukat. A már megismert **webbolt** adatbázis példáján keresztül bemutatjuk, hogy, milyen egyszerűen tudjuk ODBC-connector közbeiktatásával MS Office alkalmazások felületéről elérni és kezelni a MySQL-adatbázis adatait.

A lecke végén röviden összefoglaljuk és egy példán keresztül bemutatjuk a PHP API használatát.

Utolsó leckénk megtanulásával Ön képessé válik arra, hogy ODBC kapcsolaton keresztül manipulálja a MySQL-adatbázisokat, illetve bevezető ismeretekkel fog rendelkezni az adatbázisok adatainak webes elérésével kapcsolatban. Természetesen ez utóbbi témakör külön tantárgy tananyaga, ezért az itt látható példa távolról sem elég ahhoz, hogy webes alkalmazásokat készítsen. A tananyagban megtekinthető példákon keresztül azonban meg fogja érteni a PHP-MySQL kommunikáció alapjait.

## 11.2 A MYSQL PROTOKOLL IMPLEMENTÁCIÓI

A MySQL-szerverek szolgáltatásait különböző szoftver eszközök segítségével érhetik el a kliens oldali alkalmazások. Ezen eszközök mindegyike, a kommunikáció szabályait leíró, a TCP/IP-architektúra alkalmazási rétegébe tartozó, **MySQL protokollt** használja a kliens és a DBMS közötti adatforgalomban. Az

alkalmazások a protokoll két alacsony szintű implementációira, **mysql C library**-re (mysqllib), és az úgynevezett **natív driver**-ekre épülhetnek.

A mysql library egy C nyelven írt függvénykönyvtár. Egységes felületen teszi lehetővé különböző alkalmazások és programozási nyelvek számára a MySQL-protokoll használatát, azonban teljesítmény tekintetében elmaradhat a natív drivektól. A natív driver a mysql protokoll egy bizonyos programozási nyelvbe, vagy egy alkalmazásba beépített, annak részét képező megvalósítása.

A connectorok és API-k (Application Programming Interface) a mysql library-ra, vagy beépített natív driverre épülhetnek. A connectorok, amelyek megnevezésére a továbbiakban a konnektor szót használjuk, más adatkezelő protokollok és a MySQL-protokoll összekapcsolását teszik lehetővé. Ezzel kész alkalmazások által ismert, szabványos adatkezelő felületet biztosítanak a MySQL-adatbázisok eléréséhez.

Az API-k különböző programozási nyelveken megírt eljárások és osztályok, illetve azok metódusai, amelyek hívásával az adott programozási nyelven írt alkalmazás képes kommunikálni a MySQL-szerverrel.

Míg a konnektorok kész alkalmazások számára is lehetővé teszik a MySQL-adatbázisok adatainak elérését, addig az API-k különböző nyelveken írt adatbázis-alkalmazások készítését teszik lehetővé.

A leckénkben kiemelten foglalkozunk az ODBC-konnektorral. Az ODBC (Open DataBase Connectivity) a Microsoft által kifejlesztett külső interface, amely egységes kezelőfelületet biztosít az ODBC-t „ismerő” alkalmazások és szerverek között. Az ODBC-konnektor a MySQL adatbázisok ODBC-elérést biztosítja. Segítségével bármilyen ODBC-kompatibilis relációs adatbázis-kezelő rendszer, vagy más külső alkalmazás is hozzáférhet a MySQL-adatbázisokhoz.

Az API-k között a PHP API-t, kell feltétlenül megemlítenünk. A mysqllib könyvtárra alapuló API segítségével, PHP nyelven készíthetünk adatbázis alkalmazásokat. Mint tudjuk, a napjainkban egyre szélesebb körben használt webalkalmazások jelentős hányada készül LAMP (Linux-Apache-MySQL-PHP), illetve WAMP (Windows-Apache-MySQL-PHP) környezetben. Mindkét esetben a Windows, vagy Linux operációs rendszereken működő Apache web szerver teszi lehetővé, a MySQL adatbázisokhoz kapcsolódó, PHP nyelven megírt adatbázis alkalmazások használatát.

A mysqllib és a különböző, natív driverek, konnektorok, letölthetők a MySQL fejlesztői portáljáról, az alábbi címekről:



<http://dev.mysql.com/downloads/connector/>

<http://dev.mysql.com/downloads/connector/c/>

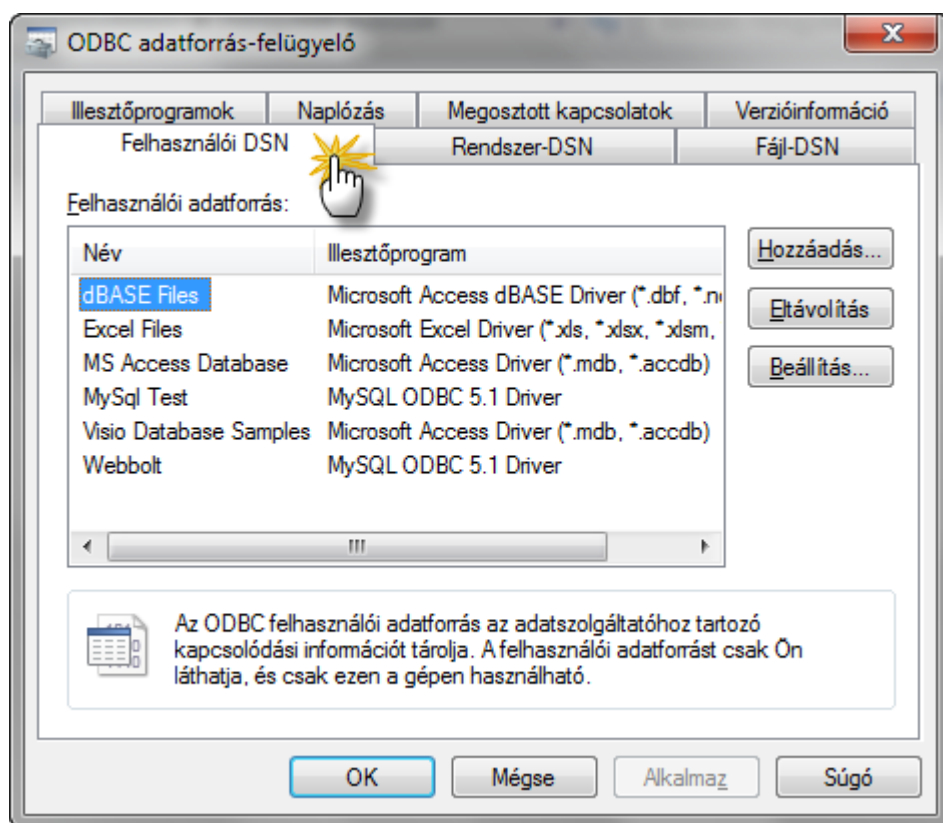
#### *5. link Natív driverek és konnektorok letöltése*

### 11.3 MYSQL ODBC CONNECTOR HASZNÁLATA

A MySQL Windows operációs rendszerre írt ODBC-konnektora, letöltés után a telepítő varázsló utasításainak követésével egyszerűen telepíthető.

A telepített konnektor bármelyik MySQL-szerver és -adatbázis elérésére alkalmas, azonban minden egyes adatbázis-kapcsolathoz külön-külön konfiguráció, úgynevezett **adatforrás létrehozása szükséges**. Az adatforrás konfigurálásakor adjuk meg az adatbázis eléréséhez és kezeléséhez szükséges adatokat, a **szerver címét** (IP|domain), **portját**, a **felhasználó nevét** és **jelszavát**, a kezelni kívánt **adatbázis** nevét, és néhány opcionális paramétert.

Az adatforrások Windows operációs rendszerek alatt a **Vezérlőpult\Felügyeleti eszközök\Adatforrások (ODBC)** alkalmazással hozhatók létre illetve konfigurálhatók. Létrehozásakor célszerű a Felhasználói adatforrások fület választva, csak saját felhasználói fiókunkból elérhető adatforrást kialakítani.

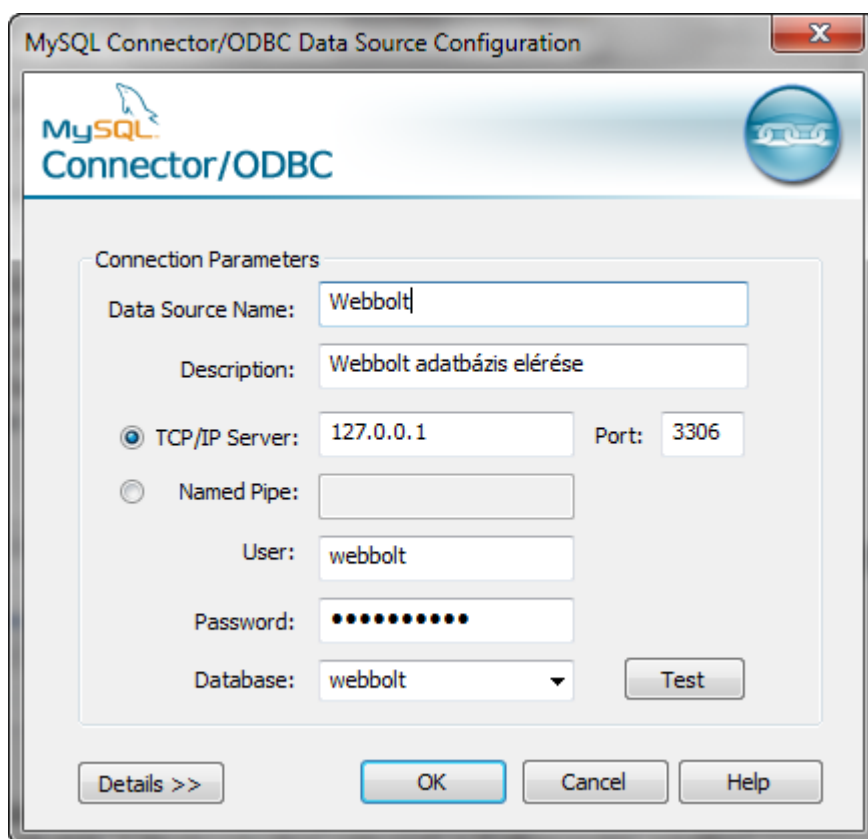


44. ábra ODBC adatforrás-felügyelő

A rendszer adatforrásokhoz számítógépünk minden felhasználója hozzáférhet. Ez alkalmat ad az adatforrásban tárolt hozzáféréssel való visszaélésre.

Új adatforrás létrehozásakor ki kell választani a megfelelő drivert. A MySQL ODBC Driver esetén az alábbi párbeszédablakban adhatjuk meg a kapcsolat paramétereit.





45. ábra Webbolt adatforrás létrehozása

A fenti ábrán a **webbolt** adatbázis eléréséhez szükséges adatforrás konfigurációja látható. Az adatforrás nevét célszerű megjegyeznünk, mert alkalmazás csatlakoztatásakor erre szükségünk lesz.



A példában a MySQL-szerver a localhoston található.

## 11.4 MS ACCESS ADATBÁZIS KAPCSOLÁSA ODBC-ADATFORRÁSHOZ

Miután az ODBC-adatforrást létrehoztuk, azon keresztül bármilyen ODBC kompatibilis alkalmazást összekapcsolhatunk az adatforrásban megadott adatbázissal. Ilyen ODBC-kompatibilis alkalmazás a MS Access, az Excel, de a mindenki által ismert MS Word is.

Érdemes megjegyezni, hogy még az Access képes olvasni és írni is az adatbázis adatit, addig az Excel és Word csak olvasni tudja azokat.

Az MS Accessben bármely létező adatbázishoz **csatolhatjuk**, vagy abba **importálhatjuk** a MySQL adatbázisok tábláit. Importáláskor az Access létrehozza a megfelelő táblákat, amelyek tartalmát a MySQL-adatbázisból másolja be. Az importálás után kapott táblák az Access adatbázisban tárolódnak, a rekordok változtatása nem kerül vissza az eredeti MySQL-táblába.

Ha a MySQL-táblákat csatoljuk, akkor azok változatlanul a MySQL-adatbázisban tárolódnak, de néhány funkciótól eltekintve pontosan úgy lesznek kezelhetők az Access felületén, mint a többi Access-tábla. Az importálással ellentétben az elvégzett adatváltoztatások frissítik a MySQL-adatbázist. Ezzel az Access a MySQL egyfajta grafikus kliensévé, az Access felületén készített alkalmazás pedig MySQL-adatbázis alkalmazásként működhet.

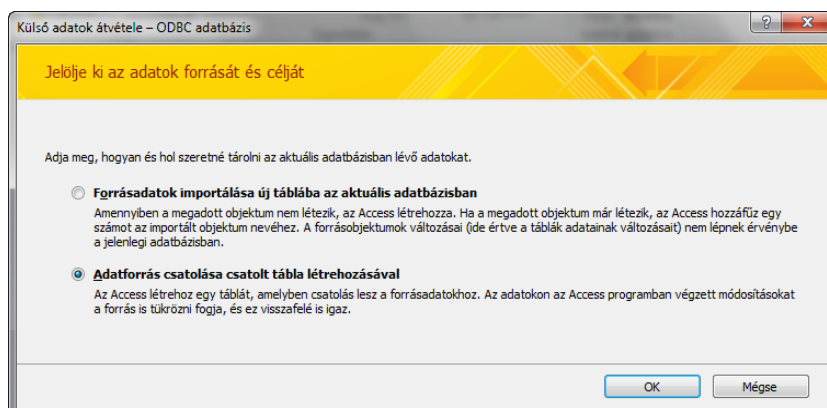
A kapcsolat kialakításához, megnyitott Access adatbázis szükséges, amely lehet „üres”, de tartalmazhat Access-objektumokat is.

A csatolás és importálás hasonló módon, az Access menüszalagjának **Külső adatforrások** lapján, az **Importálás és csatolás** csoport **ODBC adatbázis** parancsával indítható.



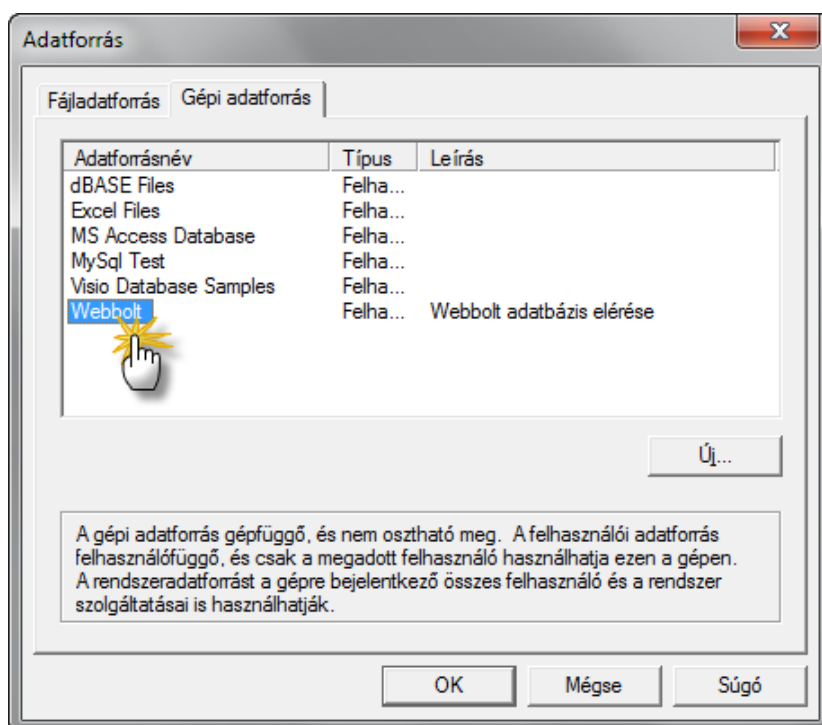
46. ábra ODBC-adatforrások elérése

A kapcsolat fölépítésére használható varázsló első lapján választhatunk az importálás és a csatolás között.



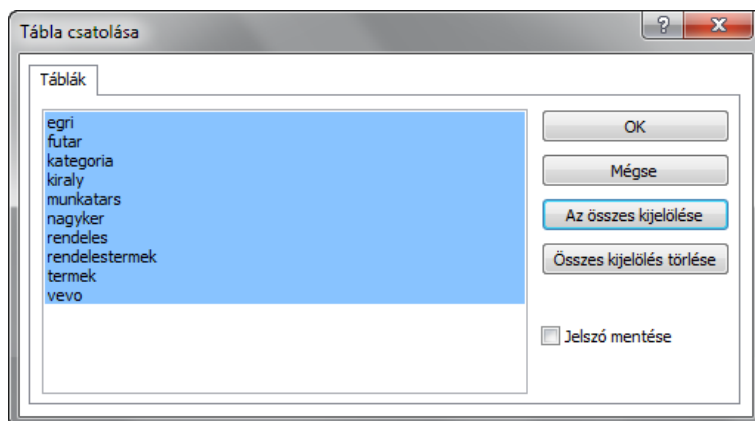
47. ábra Csatolás

A következő lépésben a korábban létrehozott adatforrások között választ-hatunk.



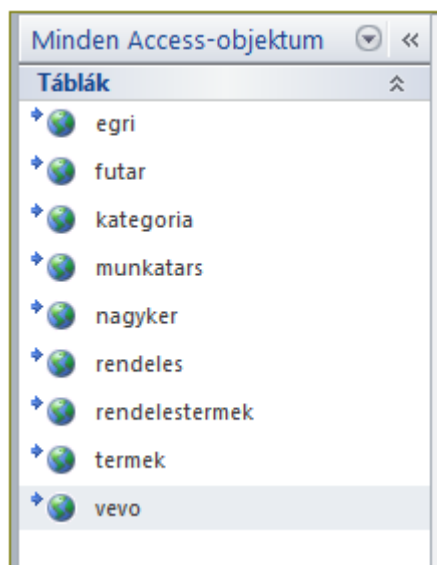
48. ábra Adatforrás kiválasztása

Ezt követően a MySQL-adatbázis megjelenő táblái és nézetei közül választjuk ki csatolni vagy importálni kívánt objektumokat.



49. ábra Csatolásra kijelölt MySQL-táblák és nézetek

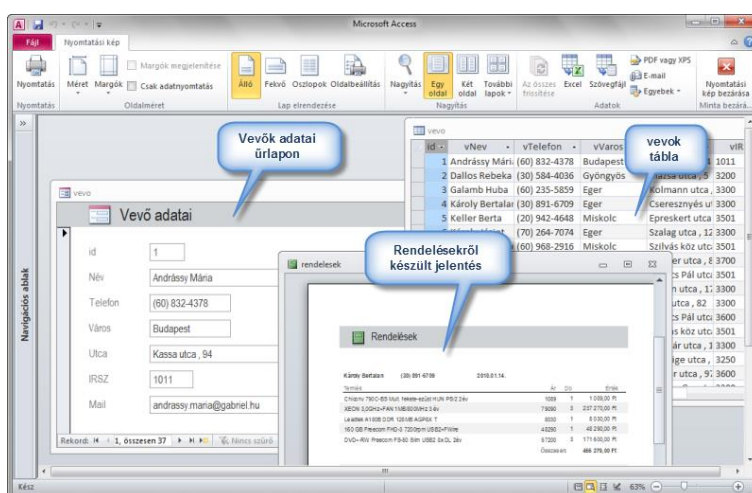
A csatolás befejezése után a navigációs ablakban speciális ikonok jelzik a csatolt objektumokat, amelyekkel szinte mindent megtehetünk, amire az Access adatbázis „saját” tábláiban is lehetőségünk van.



50. ábra Csatolt táblák az Access navigációs ablakában

Szinte mindent. A táblák megnyithatók, az azokban lévő rekordok rendezhetők, szűrhetők, kereshetők az Access grafikus felületén. Az adatokat megváltoztathatjuk, a változás pedig azonnal visszakerül a MySQL-adatbázisba is. A csatolt táblák alapján lekérdezések, űrlapok jelentések készíthetők. Valójában egyetlen dolgot nem tehetünk meg. Érthető módon nincs lehetőség arra, hogy az Access felületén megváltoztassuk a táblák szerkezetét.

Az alábbi ábra a **webbolt** adatbázis **vevo** táblájának adatlap nézetét, a tábla lapján készült űrlapot, és a **vevo-rendeles-rendelesstermek-termek-kategoria** táblák kapcsolatára épített lekérdezésen alapuló, jelentést ábrázol.



51. ábra A webbolt adatbázis kezelése az MS Access felületén

Az alábbi videofelvétel bemutatja az ODBC connector telepítését, az ODBC-adatforrás létrehozását és használatát.

## 11.5 PHP MYSQL API

Az eredeti PHP MySQL API valójában nem más, mint egy MySQL adatbázis-ok elérését és kezelését biztosító PHP függvénycsomag. Ennek továbbfejlesztésével, készült el a **mysqli** API (MySQL Improved API), amely elsősorban objektum-orientált felületében, de számos új lehetőségben is különbözik elődjétől.

A következőkben a **webbolt** adatbázis termékeinek megjelenítésén keresztül mutatunk be a MySQL eredeti API-jának használatát.

### 11.5.1 MySQL API használata

A MySQL API számos függvényt tartalmaz, amelyekkel az adatbázisok kezeléséhez kapcsolódó, különböző funkciók valósíthatók meg.

A függvények visszatérési értéke általában az elvégzett művelet eredménye, illetve lebonyolítás sikeres voltáról tájékoztató logikai érték. A függvények által visszaadott eredményt vizsgálatával eldönthető, hogy sikeres volt-e a hívás. Ha igen az eredményt föl kell dolgozni, ha nem, akkor valamilyen módon kezelni kell a hibát.

```
\
\
\
\
\
\
\
\
\
\
\
```

A hibakezelés a PHP-programozás önálló fejezete, ezért az itt bemutatott példákban a legegyszerűbb módszert, a program hibaüzenet mellett történő leállítását használjuk.

Az egyszerű kapcsolódás, az adatok lekérdezése és megjelenítése az alábbi lépésekben valósítható meg:

- Kapcsolat fölépítése a program és a MySQL-adatbázis között.
- Adatbázis kiválasztása.
- Lekérdezés elküldése.
- Lekérdezés eredményhalmazának feldolgozása.
- Kapcsolat lezárása.
- Weblap megjelenítése.

A kapcsolat fölépítését a **`mysql_connect()`** függvénnyel végezhetjük el. A függvény három fontos paramétert, az adatbázis-**szerver címét**, a **felhasználó nevét**, és **jelszavát** várja. Visszatérési értéke sikeres kapcsolódás esetén az úgynevezett **kapcsolatazonosító**, amellyel a továbbiakban, a lezárásáig hivatkozhatunk a kapcsolatra. Ha a kapcsolat sikertelen a visszatérési érték **false** (logikai hamis). A kapcsolat lezárásához a paraméter nélküli **`mysql_close()`** függvényt használjuk

```
<?php
    //Szükséges változók deklarálása
    $host = "localhost";
    $user = "webbolt";
    $passwd = "netuddmeg";
    $database="webbolt";
    $sql="SELECT * FROM termek t NATURAL JOIN
kategoria k
                ORDER BY k.kNev, t.tMegnevezes;";

    //Kapcsolat fölépítése
    $connid = mysql_connect($host, $user, $passwd)
                or die("A szerver nem elérhető");
    //Ide kerül a további kód
    echo("A kapcsolat létrejött");
    mysql_close();
?>
```

A kapcsolat fölépítése, a kapcsolatazonosító lekérdezése az alábbi sorral történik:

```
$connid = mysql_connect($host, $user, $passwd) or
die("A szerver nem elérhető");
```

Sikeres kapcsolódás esetén az azonosító a **\$connid** változóba kerül. A példában a hibakezelés egyszerű, de egyben primitív módszeréről használjuk. A függvényhívás után, de vele egy programsorban található **or die("A szerver nem elérhető")** parancs akkor lép működésbe, ha a függvény **false** (hamis) eredményt ad vissza. A **die()** megszakítja a programot, és a zárójelben lévő szöveget küldi a kimenetre.

```
<?php
//A MySQL adatbázisból lekérdezett adatok megjelenítésekor fontos,
//hogy a DBMS, a weblapon használt karakterkódolásnak megfelelően
//küldje el a lekérdezett mezőértékeket. Ezt a
mysql_set_charset(kódolás); függvény hívásával érhetjük el.
Jelen esetben utf-8 kódolást használunk:
mysql_set_charset("utf8");
```

A sikeres kapcsolat esetén a **mysql\_select\_db()** függvénnyel választható ki a szükséges adatbázis. A függvény paramétere az adatbázis neve, visszatérési értéke siker esetén **true**, ellenkező esetben **false**.

```
$ret=mysql_select_db($database) or die("Az adatbázis  
nem található");
```

Ha az adatbázis kiválasztása is sikeres volt, a **mysql\_query()** függvény-  
nyel küldhetjük el a DBMS-nek a paraméterként megadott SQL-mondatot. Sike-  
res lekérdezés esetén az **eredményhalmaz**, hiba esetén **false** értéket ka-  
punk eredményként.

```
$res=mysql_query($sql) or die("A lekérdezés nem  
futtatható");
```

Az eredményhalmazt többféle módszerrel is feldolgozhatjuk a PHP adattí-  
pusai és vezérlési szerkezetei segítségével. Az egyik lehetőség a  
**mysql\_fetch\_assoc()** függvény használata. A függvény paramétere az  
eredményhalmazt tartalmazó változó, eredménye pedig következő rekordjából  
létrehozott asszociatív tömb. A tömbelemek értékei a mezőértékek, nevei a  
mezőnevek.

A függvény többször is meghívható, és mindig a következő rekorddal tér  
vissza. Az utolsó rekord utáni hívás **false** eredményt ad.

```
//Eredményhalmaz feldolgozása  
while($row=mysql_fetch_assoc($res)) {  
  
    echo($row['tMegnevezes'] . " . . . Ár: " . $row['tEladAr'] . "<b  
r>");  
}
```

Rövidke példaprogramunk teljes kódja a következő:



```
<?php
```



```
$host = "localhost";
```



```
$user = "webbolt";
```



```
$passwd = "netuddmeg";
```





```
$database="webbolt";
```



```
$sql="SELECT * FROM termek t NATURAL JOIN kat-  
egoria k
```



```
ORDER BY k.kNev,  
t.tMegnevezes;";
```



```
//Kapcsolat fölépítése
```



```
$connid = mysql_connect($host, $user, $passwd)  
or die("A szerver nem elérhető");
```



```
mysql_set_charset("utf8");
```



```
//Adatbázis kiválasztása
```



```
$ret=mysql_select_db($database) or die("Az  
adatbázis nem található");
```



```
//További kód...
```



```
$res=mysql_query($sql) or die("A lekérdezés nem  
futtatható");
```



```
//Eredményhalmaz feldolgozása
```



```
echo("<h1  
align='center'>Árlista</h1><br><br>");
```



```
echo("<table align='center' border='1'>");
```



```
echo("<th>Kategória</th><th>Termék</th><th>Ár</th>
");
```



```
while($row=mysql_fetch_assoc($res)){
```



```
echo("<tr>");
```



```
$prod=$row['tMegnevezes'];
```



```
$categ=$row['kNev'];
```



```
$cost=$row['tEladAr'];
```



```
if($lastcateg!=$categ){
```



```
$lastcateg=$categ;
```



```
} else {
```



```
$categ="&nbsp;";
```



```
}
```



```
echo("<td>{$categ}</td><td>{$prod}</td><td>
align='right'>{$cost}</td>");
```



```
echo("</tr>");
```



```
}
```



```
echo("</table>");
```



```
//Kapcsolat lezárása
```



```
mysql_close();
```



```
?>
```

## 11.6 ÖSSZEFOGLALÁS, KÉRDÉSEK

### 11.6.1 Összefoglalás

Utolsó leckénkben a MySQL-adatbázisok külső alkalmazásokkal történő összekapcsolásáról tanultunk. Megismertük a mysql protokoll, mysql C library, a natív driver, a connector és API fogalmakat. Megtanultuk, hogy konfigurálható a MySQL ODBC Connectora, amely ODBC kompatibilis alkalmazások számára teszi lehetővé a MySQL adatbázisok elérését. A lecke példáján keresztül láttuk, hogyan csatolhatók a MySQL-táblák, tetszőleges MS Access adatbázisokba, hogyan válhat az Access, a MySQL grafikus kliensévé, és hogyan tehetjük MySQL-alkalmazássá az Access segítségével készített adatbázis-kezelő felületet. Leckénk utolsó felében, rövid példában ismeretük meg a PHP MySQL API-jának használatát.

### 11.6.2 Önellenőrző kérdések

1. Mit az a mysql protokoll?

- A MySQL-protokoll a MySQL szerverek és kliens oldali alkalmazások kommunikációjának protokollja, ami a TCP/IP-architektúra legfelső, alkalmazási rétegébe tartozik.

2. Mik azok a MySQL API-k?

- A MySQL API-k az egyes programozási nyelvek eljárások, függvények, illetve osztályok és metódusok formájában létrehozott kiegészítései. Segítségükkel MySQL adatbázis alkalmazások hozhatók létre az adott programozási nyelv eszközeivel.

3. Hogyan kezelheti a MySQL adatbázisokat az MS Access felületén?

- A MySQL adatbázisok és az Access kapcsolatát a MySQL ODBC Connector segítségével valósíthatjuk meg. A megfelelő adatforrás létrehozása után az Access-adatbázisaiba csatolhatjuk, vagy importálhatjuk a MySQL-adatbázis tábláit és nézeteit.

4. Mi a különbség táblák csatolása és importálása között?

- Importálás után az adatok az Access adatbázis részévé válnak. Az Access-ben elvégzett adatváltoztatások nem kerülnek többé vissza a MySQL-adatbázisba. Csatoláskor azonban az Access felületén kezelhető táblák, és nézetek változatlanul a MySQL-szerveren tárolódnak. Az Accessben elvégzett változtatások a MySQL-adatbázis frissítését eredményezik.

5. Mit csinál az alábbi PHP program?

```
<?php
    $host = "localhost";
    $user = "webbolt";
    $passwd = "netuddmeg";
    $database="webbolt";
    $sql="SELECT vNev FROM vevo v
                                ORDER BY v.vNev;";
    $connid = mysql_connect($host, $user,
    $passwd) or die("A szerver nem elérhető\n");
    $ret= mysql_select_db($database) or
    die("A(z) {$database} adatbázis nem találha-
    tó\n");
    $res= mysql_query($sql) or die("A {$sql}
    lekérdezés nem futtatható\n");
    while($row=mysql_fetch_assoc($res)){
        echo ($row[vNev] . "<br>");
    }
    echo("</table>");
    mysql_close();
?>
```

- Névsorban listázza a webbolt adatbázis vevok táblájában tárolt vevők neveit.

# 12. ÖSSZEFOGLALÁS

## 12.1 TARTALMI ÖSSZEFOGLALÁS

Tananyagunk végén tekintsük át röviden az egyes leckékben megszerezhető ismereteket és kompetenciákat!

### 12.1.1 Bevezetés

Bevezető leckénkben megismerhette a tananyag fölépítését, a lecek szerkezetét, tanulási tanácsokat kapott, és megtudhatta hogyan férhet hozzá a tananyagban használt forrásokhoz.

### 12.1.2 MySQL-adatbázisok kezelése

A MySQL-adatbázisok kezelése című lecke összefoglalta azokat az ismereteket, amelyekre szükség van tananyagunk elsajátításához. Itt esett szó az relációs adatmodellhez kapcsolódó fogalmakról, az adatbázis-rendszerekről, az SQL nyelvről, a MySQL DBMS-ről és a kliens használatáról.

### 12.1.3 Választó lekérdezések készítése

A 3. leckében tanultuk meg a SELECT paranccsal készíthető választó lekérdezések használatát. Itt tanulhatta meg, mit értünk rekordforrás, forráshalmaz, és eredményhalmaz alatt. Ebben a leckében esett szó mező meghatározásokról, számított mezőkről, a FROM, a WHERE, az ORDER BY és a LIMIT záradékokról.

### 12.1.4 Kapcsolódó táblák rekordjainak kezelése

Ebben a leckében tekintettük át a lekérdezés készítés egyik legizgalmasabb területét, a többtáblás rekordforrások kialakítását. Olvashatott a leckében a kapcsolatok nyelvi megfogalmazásának lehetőségeiről, a szoros és laza illesztésekről, és a rekordforrás tábláihoz kapcsolt álnevek használatáról.

### 12.1.5 Függvények használata az adatkezelésben

Az 5. lecke a számított értékek előállításában hasznos függvényekkel foglalkozott. A leckében megismerte a függvények használatával kapcsolatos olyan általános tudnivalókat, mint a hívás, visszatérési érték, típus, formális és aktuális paraméterlista. Három kategóriába sorolva áttekintettük a legfontosabb függvényeket, amelyek a MySQL függvénytárának csupán töredékét alkotják.

### 12.1.6 Statisztikai számítások SQL-ben

Ebben a leckében olvashatott statisztikai számítások megvalósítására alkalmas összesítő lekérdezésekről. Megismerhette az összesítő függvényeket és használatukat, a rekordok részhalmazainak kialakítására alkalmas GROUP BY és az összegzett mezők szűrésére használható HAVING záradékokat.

### 12.1.7 Allekérdezések használata

Az Allekérdezések használata című leckében azt tanulhatta meg, hogyan biztosíthat adatokat egy lekérdezés számára egy másik lekérdezés segítségével. Megtanulhatta az allekérdezések beágyazásának nyelvtani, és logikai szabályait, és a megismerhette az allekérdezések lehetséges felhasználási módjait.

### 12.1.8 Rekordok beszúrása az adatbázis tábláiba

A 8. leckével a DQL-lekérdezések után áttértünk a DML-mondatok készítésére. Ebben a leckében az INSERT parancs használatával ismerkedtünk meg. Megtanultuk, hogyan lehet egy rekord összes mezőértékét, vagy csak kiválasztott mezők adatait beszúrni egy táblába. Olvashatott az alapértelmezett, a NULL, és az AUTO\_INCREMENT értékek kezeléséről. Megtanulhatta, hogyan olvashat be az adatbázis tábláiba külső szövegfájlokban tárolt rekordokat.

### 12.1.9 Rekordok módosítása, és eltávolítása

Ebben a leckében a tárolt rekordok módosításáról, és törléséről tanultunk. A 7. lecke anyagára visszatekintve megtanulgatta, hogyan lehet a DML-műveletekben allekérdezésekkel kiválogatni a feldolgozott rekordokat.

### 12.1.10 A MySQL továbbfejlesztett lehetőségei

A továbbfejlesztett lehetőségeket tárgyaló leckében a tranzakciókezelésről, a tárolt eljárások és függvények deklarálásáról és használatáról, valamint a triggerokról tanultunk.

### 12.1.11 Adatok megjelenítése különböző platformokon

Tananyagunk utolsó előtt lelkéjében az MySQL ODBC Connector és a PHP MySQL API példáján keresztül ismertük meg a mysql protokoll, a mysqllibrary, a natív driver, a connector és az API fogalmakat.

A lecke bemutatta a MySQL-adatbázisok ODBC-adatforrásként történő felhasználását, és rövid példában mutatta be a PHP MySQL API használatát.

### **12.1.12 Összefoglalás**

Utolsó leckénkben, a tananyagra visszatekintve összegeztük a leckék át-tanulmányozásával megszerezhető ismereteket.

## **12.2 FOGALMAK**

### **12.2.1 Bevezetés**

### **12.2.2 MySQL-adatbázisok kezelése**

adat, információ, kommunikáció, adatbázis-rendszer, adatbázis-kezelő rendszer, adatmodell, SQL, MySQL

### **12.2.3 Választó lekérdezések készítése**

SELECT, eredményhalmaz, álnév, számított mező, WHERE, LIMIT, ORDER BY, DISTINCT

### **12.2.4 Kapcsolódó táblák rekordjainak kezelése**

Illesztés, álnevek, INNER JOIN, OUTER JOIN, Implicit JOIN, NATURAL JOIN

### **12.2.5 Függvények használata az adatkezelésben**

Függvény, szövegkezelés, dátum és időkezelés, matematikai függvények

### **12.2.6 Statisztikai számítások SQL-ben**

összegző lekérdezés, összesítő függvény, GROUP BY, HAVING, WITH ROLLUP

### **12.2.7 Allekérdezések használata**

Allekérdezés, allekérdezés mező meghatározásban, allekérdezés rekord-forrásában, allekérdezés WHERE záradékban

### **12.2.8 Rekordok beszúrása az adatbázis tábláiba**

INSERT, VALUES, SET, DEFAULT, NULL, INSERT opcionális elemei, LAST\_INSERT\_ID(), SELECT ...INTO OUTFILE

### **12.2.9 Rekordok módosítása, és eltávolítása**

Táblák mentése, UPDATE, DELETE, USING, TRUNCARE

### **12.2.10 A MySql továbbfejlesztett lehetőségei**

tranzakció, BEGIN, START, COMMIT, ROLLBACK, VIEW, PROCEDURE, FUNCTION, CALL, TRIGGER

### **12.2.11 : MySQL DBMS és adatbázis alkalmazások kommunikációja**

protokoll, connector, API, MySQL ODBC connector, PHP MySQL API

## **12.3 ZÁRÁS**

Tananyagunk végéhez érve megköszönjük a kitartó érdeklődését, és kifejezzük abbéli reményünket, hogy a lelkékben olvasott adatok információvá, azok pedig jövőbeni munkáját eredményessé, hatékonyá tévő tudássá alakultak az olvasóban.



# 13. KIEGÉSZÍTÉSEK

## 13.1 IRODALOMJEGYZÉK

### 13.1.1 Hivatkozások

#### *Könyv*

CELKO J.: *SQL felsőfokon*. Budapest, Kiskapu kiadó, 2002.

HALASSY B.: *Az adatbázis-tervezés alapjai és titkai*. Budapest, IDG Magyarországi Lapkiadó Kft., 1994

MELONI J.C.: *Tanuljuk meg a MySQL használatát 24 óra alatt*. Budapest, Kiskapu kiadó, 2003.

SCHWARTZ B., ZAITSEV P., TKACHENKO V.: *High Performance MySQL*, O'Reilly Media, Inc, 2012.

#### *Elektronikus dokumentumok / források*

ORACLE TEAM: *MySQL documentation*, Oracle, [online] [2012.06.01]

< <http://dev.mysql.com/doc/> >