# M. Fejezet

# MiniSQL adatbázisok elérése

Ebben a részben röviden bemutatunk egy, a MiniSQL adatbázisok elérésére használatos eszközt. A MiniSQL egy egyszerű, olcsó adatbázis-kezelő[1]. A MiniSQL szerver az ANSI SQL egy viszonylag szűk, de már komolyabb alkalmazásokban is használható részhalmazát támogatja[2], és a kliens/szerver modell keretei között lehetővé teszi azt is, hogy akár hálózaton keresztül érjünk el adatbázist. Mivel a MiniSQL-hez könnyen hozzájuthatunk, ezért sokan használják weben is elérhető adatok tárolására, ezért érdemes megnézni egy egyszerű példaprogram tanulmányozásával ennek a kapcsolatnak a gyakorlati alkalmazhatóságát. A fejezetben mellékelt példaprogram segítségével ezenkívül tovább mélyíthetjük a JDBC-ről megszerzett ismereteinket is.

Az adatbázis eléréséhez az Imaginary által kifejlesztett MiniSQL JDBC meghajtót fogjuk használni (letölthető: `http://www.imaginary.com/Java`). Ez a JDBC meghajtó Java nyelven készült, ezért minden Java környezetben jól használható. Alkalmazásánál természetesen figyelembe kell venni a MiniSQL korlátait is, mivel a MiniSQL-ből hiányzó lehetőségek ezzel az adatbázis-meghajtóval sem elérhetők (ilyen lehetőségek például a tranzakciókezelés, a nézettáblák definiálhatósága stb.), használatukkor a futtató rendszer a hibának megfelelő kivételt fog generálni.

Felhívjuk a figyelmet arra, hogy ezen fejezet célja nem a MiniSQL adatbázis-kezelő ismertetése. Sem az általa elfogadott SQL nyelvet, sem az adatbázis-kezelő telepítését nem fogjuk ismertetni, ugyanis ezekről részletes ismertetők érhetők el a MiniSQL rendszerrel szállított dokumentációkban. A célunk a továbbiakban az lesz, hogy lehetővé tegyük egy már telepített MiniSQL adatbázis-kezelő elérését Java programokból.

Először tekintsük át egy MiniSQL adatbázis konfigurációs fájljának egy részletét, amely az adatbázis elérhetőségi paramétereit írja le (a fájl neve a legtöbb rendszeren `msql.conf`). A fájlrészlet a következő:

```
Inst_Dir = /usr/local/msql
mSQL_User = msql
Admin_User = root
TCP_Port = 1114
```

A fentiekből kiderül, hogy a MiniSQL adatbázis-kezelőt melyik könyvtárba telepítették, kiderül az is, hogy mely felhasználó jogaival működik az adatbázis-kezelő szerver, és az is kiderül, hogy melyik felhasználó az adatbázis-kezelő adminisztrátora (aki például jogosult új adatbázisok létrehozására; a MiniSQL adatbázis-kezelő képes több adatbázis egyidejű kezelésére is: mindegyik adatbázisban egymástól függetlenül hozhatunk létre adatokat tartalmazó táblákat, indexeket stb.). A negyedik sorból pedig azt láthatjuk, hogy az adatbázis-kezelőt az 1114-es TCP porton érhetjük el; erre az információra szükségünk lesz a Java kapcsolat megteremtésekor.

---

[1] non-profit szervezetek, illetve oktatási intézmények számára a MiniSQL használata e sorok írásakor ingyenes - a konkrét licenszelési feltételeket meg kell nézni a szoftverrel szállított, illetve hálózatról letöltött dokumentumokban; alkalmazhatóságának jogi feltételeinek mérlegelésekor nem szabad erre a bekezdésre hagyatkozni!

[2] Az SQL nyelvről magyar nyelvű leírást [SQLh]-ban találhatunk, míg a MiniSQL-re vonatkozó részletekről [MSQLspec]-ben olvashatunk.

## M.1. Egy példaadatbázis létrehozása

A fejezetben bemutatott példaprogramok futásához szükségünk lesz egy adatbázisra, melynek neve: `Adatok`. Természetesen - a példaprogramok módosításával - használhatnánk bármilyen másik adatbázist is, de azért, hogy a saját adataink ne keveredjenek a következőkben beírt tesztadatokkal, hozzuk létre ezt az adatbázist. Egy új adatbázis létrehozását - a fenti konfigurációs fájl alapján - `root` nevű felhasználóként végezhetjük el a következő parancs beadásával (az adatbázis neve: `Adatok`):

```
msqladmin create Adatok
```

Ezután a létrehozott adatbázist feltölthetjük adatokkal. Ehhez használhatjuk például a következő UNIX/Linux Bourne Shell scriptet (megjegyezzük, hogy ehhez már nem kell `root` néven bejelentkeznünk):

```
msql Adatok <<VEGE
DROP TABLE Szemelyek
\g
DROP TABLE Alkalmazottak
\g
CREATE TABLE Szemelyek (Nev CHAR(25) NOT NULL, Cim CHAR(25),
                        Neme CHAR(1), Eletkor INT NOT NULL)
\g
CREATE TABLE Alkalmazottak (Nev CHAR(25) NOT NULL, Beosztas CHAR(25))
\g
INSERT INTO Szemelyek VALUES ('Barna Farkas', 'Kerekerdo 21', 'F', 23)
\g
INSERT INTO Szemelyek VALUES ('Kapatos Miska', 'Hecc ut 45', 'F', 45)
\g
INSERT INTO Szemelyek VALUES ('Rozsa Roxana', 'Hargita ut 92', 'N', 34)
\g
INSERT INTO Szemelyek VALUES ('Voros Maria', 'Sajt setany 42', 'N', 32)
\g
INSERT INTO Szemelyek VALUES ('Szarka Ilona', 'Arany ter 4', 'N', 25)
\g
VEGE
```

Vagyis a fenti példában az `msql` parancsot megadva az `Adatok` adatbázist érjük el. Az `msql` program a MiniSQL legegyszerűbb parancsértelmezős felülete. \g sorral lezárt SQL utasításokat adhatunk meg neki a szabványos bemenetről, majd ezeket a parancsokat átadja az adatbázis-kezelő szervernek, amely végrehajtja azokat.

A fenti példában az `msql` parancs az általa végrehajtandó SQL utasításokat a billentyűzet helyett a rákövetkező sorokban található SQL utasításokból veszi, egészen addig, amíg egy `VEGE` kezdetű sort nem talál (ez a Bourne shell egyik számunkra kényelmesen használható átirányítási tulajdonsága).

Látható, hogy a fenti példában létrehoztunk egy `Szemelyek` és egy `Alkalmazottak` nevű táblát (az előbbibe felvettünk néhány sort, míg az utóbbi táblát üresen hagytuk). Mielőtt létrehoznánk a nevezett táblákat, letöröljük az esetleg korábban már létrehozott azonos nevű táblákat (ez hasznos lehet, ha tiszta lappal akarunk indulni).

## M.2. Egy egyszerű Mini SQL alkalmazás

Miután elkészítettük a példaadatbázisunkat, írjunk egy egyszerű alkalmazást, amely végrehajt egy lekérdezést JDBC-n keresztül, kiírja az eredménytábla oszlopainak az ajánlott

nevét, majd kiírja a lekérdezés eredménytáblájának a sorait is. Végül kíratjuk vele az adatbázisban levő táblák neveit és típusait is (közönséges tábla vagy mondjuk nézettábla) - ez egy egyszerű példa az adatbázis adatszótárának elérésére.

Először tekintsük a példaprogram forráskódját:

```
// MSQLTest.java
//
// Teszteli a MiniSQL adatbázis-kapcsolatot.

import java.net.URL;
import java.sql.*;

class MSQLTest {
  public static void main(String argv[]) {
    try {
      Class.forName("com.imaginary.sql.msql.MsqlDriver");
      String url = "jdbc:msql://rozsika:1114/Adatok";
      Connection con = DriverManager.getConnection(url, "msql", "");
      Statement utas = con.createStatement();
      ResultSet rs = utas.executeQuery("SELECT * FROM Szemelyek ORDER BY Nev");
      ResultSetMetaData oszi = rs.getMetaData();
      System.out.println("Az eredmény oszlopainak száma:"+oszi.getColumnCount());
      for (int i=1;i<=oszi.getColumnCount();i++) {
        System.out.print(i+". oszlop ajánlott fejléce:"+oszi.getColumnName(i));
        System.out.print("  tipusa:"+oszi.getColumnTypeName(i));
        if (oszi.isNullable(i)==ResultSetMetaData.columnNoNulls) {
          System.out.print(" (NOT NULL) ");
        }
        System.out.print("\n");
      }

      System.out.println("Az eredmény sorai:");
      while(rs.next()) {
        int KorEgyevmulva = rs.getInt("Eletkor")+1;
        String KorSzovegesen = rs.getString("Eletkor");
        String Nev = rs.getString("Nev");

        System.out.print("Név: " + Nev);
        System.out.print("   Életkor: " + KorSzovegesen);
        System.out.print("   Életkor 1 év múlva: " + (KorEgyevmulva));
        System.out.print("\n");
      }
      utas.close();
      System.out.println();
      DatabaseMetaData dd = con.getMetaData();
      rs = dd.getTables(null, null, null, null);
      while( rs.next() ) {
        System.out.println("Tábla neve: " + rs.getString("TABLE_NAME") +
            "  típusa: " + rs.getString("TABLE_TYPE") );
      }
      con.close();
    }
    catch( Exception exc ) {
      System.out.println(exc.getMessage());
    }
  }
}
```

Ha a fenti programot le akarjuk fordítani, szükségünk van egy MiniSQL JDBC-meghajtóra. Ehhez a letöltött, illetve kicsomagolt installációs anyagból a JDBC-meghajtó osztályait tartalmazó - általában JAR formátumú - könyvtárfájl elérési útvonalát tegyük be a lokális osztálybetöltési útvonalakba. Ezután a fenti programot a következő paranccsal fordíthatjuk le:

```
javac MSQLTest.java
```

Majd a

```
java MSQLTest
```

paranccsal futtathatjuk.

A fenti programot lefuttatva az előbb bemutatott adatbázis környezetben a következő eredményt kapjuk:

```
Az eredmény oszlopainak száma:4
1. oszlop ajánlott fejléce:Nev  típusa:CHAR (NOT NULL)
2. oszlop ajánlott fejléce:Cim  típusa:CHAR
3. oszlop ajánlott fejléce:Neme  típusa:CHAR
4. oszlop ajánlott fejléce:Eletkor  típusa:INT (NOT NULL)
Az eredmény sorai:
Név: Barna Farkas   Életkor: 23   Életkor 1 év múlva: 24
Név: Kapatos Miska   Életkor: 45   Életkor 1 év múlva: 46
Név: Rozsa Roxana   Életkor: 34   Életkor 1 év múlva: 35
Név: Szarka Ilona   Életkor: 25   Életkor 1 év múlva: 26
Név: Voros Maria   Életkor: 32   Életkor 1 év múlva: 33

Tábla neve: Szemelyek  típusa: TABLE
Tábla neve: Alkalmazottak  típusa: TABLE
```

## M.3. Kapcsolatfelvétel MiniSQL adatbázissal

A fenti példaprogram MiniSQL-meghajtótól függő részét lényegében a következő három sor tartalmazza:

```
Class.forName("com.imaginary.sql.msql.MsqlDriver");
String url = "jdbc:msql://rozsika:1114/Adatok";
Connection con = DriverManager.getConnection(url, "msql", "");
```

Az első sor egyszerűen betölti a megfelelő - MiniSQL-hez való - JDBC meghajtót megvalósító osztályt.

A következő sorban állítottuk össze az elérni kívánt MiniSQL adatbázis elérési URL-jét: megmondjuk, hogy az elérni kívánt adatbázis egy MiniSQL adatbázis, mely a `rozsika` nevű számítógépen az 1114-es TCP porton érhető el; az elérni kívánt adatbázis neve: `Adatok`.

A harmadik sorban történik a kapcsolatfelvétel az adatbázissal. A `getConnection` metódus első paramétere az előző sorban összeállított adatbázis-leíró URL, a második paraméter a bejelentkezéshez használandó felhasználói azonosító, a harmadik paraméter pedig a második paraméterben megadott felhasználó bejelentkezési jelszava.

## M.4. Az adatbázisok és az SQL

Egy adatbázis lényegében hosszú ideig - az információkat felvevő programok élettartamánál lényegesen hosszabb ideig - tárolt információk összessége. Az adatbázisokat általában valamilyen adatbázis-kezelő rendszer kezeli. Az adatbázis-kezelő rendszerektől általában elvárjuk, hogy lehetőséget nyújtsanak felhasználóiknak új adatbázisok készítésére (megadva azok ún. sémáját, a tárolt adatok szerkezetét, valamilyen ún. adatdefiníciós eszközzel, adatdefiníciós nyelvvel). A másik fontos követelmény az adatbázis-kezelőkkel szemben a tárolt adatok hatékony és sokrétű lekérdezhetőségének a biztosítása, amely általában valamilyen adatbázis-lekérdező nyelvvel oldható meg.

Az adatbázisrendszerek története folyamán számos adatmodell alakult ki, amelyek a tárolt adatok ábrázolását más-másképp oldják meg. Ma a legelterjedtebb az ún. relációs adatmodell, ahol az adatokat ún. táblák (táblázatok) formájában tárolják. Itt a felhasználónak nem kell foglalkoznia az adatok belső tárolási módjával, ami hatékonysági és megbízhatósági szempontokat szem előtt tartva nagyon sokféleképpen megoldható (hash-függvénnyel és hash-táblákkal, keresőfákkal vagy éppen strukturálatlan adatfolyam formában). A lekérdezésre a legtöbb mai relációs adatbáziskezelő rendszer az ún. SQL nyelvet biztosítja, ezt érhetjük el Java nyelvű programjainkból is a JDBC adatbáziskapcsolatkezelő könyvtárral.

Táblára példaként láthattuk az előbbi pontok `Szemelyek` valamint Alkalmazottak tábláját, amelyek oszlopai tartalmazzák az adatbázisban tárolt egyedek jellemzőit (ún. attribútumait); az egyes egyedekről tárolt adatok pedig a tábla 1-1 sorát képezik. A `Szemelyek` tábla négy attribútumot (oszlopot) tárol minden egyes személyről, a táblában pedig minden egyes személynek pontosan egy sor felel meg. Egy sorban az első oszlop a személy nevét tartalmazza, a második oszlop az illető személy címét, a harmadik oszlop az illető személy nemét tárolja, míg a negyedik oszlop a személy életkorát tárolja (megjegyezzük, hogy a legtöbb adatbázisban az életkor helyett érdemesebb a születési időpontot tárolni, mivel az életkor az idő múlásával változhat).

Egy adatbázis struktúrájának kialakítása során számos konzisztencia-problémára figyelnünk kell. Kerülni kell a redundanciát, amely egy adat többszörös tárolását jelentheti. Ez akár módosítási problémákat is maga után vonhat, amely például úgy jelenik meg, hogy ha megváltoztatjuk az egyik sorban tárolt valamely információt, akkor ugyanaz az információ változatlanul megmarad valamely másik sorban. Az adatbázisok még említést érdemlő tervezési hibái az ún. törlési problémák körébe sorolható. Ilyenkor egy sorban túl sok információ van, amely a sortörlési műveletet durvává teszi azáltal, hogy egy sor törlésével esetleg több információt elveszthetünk az adatbázisból, mint ami ténylegesen feleslegessé, illetve törölhetővé vált. Az adatbázis-tervezés elmélete számos eszközt és normálformát ismer, amelyeket a tervezés során figyelemmel tartva olyan adatbázishoz juthatunk, amely mentes az előbb említett és más hasonló jellegű anomáliáktól (ez a tervezésre fordított idő természetesen a programozás és a program későbbi karbantartási fázisában időmegtakarítást eredményez, ami miatt egyértelműen azt mondhatjuk, hogy ez egy kifizetődő ráfordítás).

A táblák lehetnek függetlenek, illetve egyes táblák közt bizonyos függőségeket is megállapíthatunk. A független tábla elnevezés alatt azokat a táblákat értjük, amelyek sorai nem tartalmaznak olyan adatokat, amelyek csak más táblában tárolt adatokkal együtt nyernek értelmet. A függőségeket tartalmazó táblák általában más táblákkal együttesen szolgáltatják a programozó (ill. más adatfeldolgozó) számára szükséges információkat. A funkcionális függőségek tárgyalásával nem foglalkozunk, mivel az adatbázis-elmélettel foglalkozó könyvek bő információkat tartalmaznak erről a témakörről. A függőségek általában vagy azon alapulnak, hogy az adatbázis egy táblájának sorai egyértelmű objektumokat reprezentálnak (például minden sor egy személy adatait tartalmazza, a személy egyedi adószámával együtt), vagy pedig a valós világ üzleti folyamatainak modellezése-

képp jöttek létre (például minden személy évente legfeljebb egy személyi jövedelemadó-bevallást nyújt be az adóhatósághoz feldolgozásra).

A táblák szerkezetét, a függőségeket (az adatbázis sémáját) gyakran szokás szemléletesen grafikus módon ábrázolni. Erre egy egyszerű eszköz az egyed-kapcsolat diagram. Ez a következő komponensekből építi fel a modellt:

- Egyedhalmazokból, amelyek általában a valós világ valamely osztályainak elemeit ábrázolják (lásd az objektum-orientált tervezés osztálykialakítási szempontjait, amely itt is alkalmazható a valamilyen szempontból hasonló tulajdonságú egyedekből történő absztrahálásra). Az egyes egyedek tulajdonságait azok attribútumaival jellemezzük. Az attribútumok rögzítik az egyedek tulajdonságainál - azok leírásánál - felhasználható lehetséges értékek halmazát, azok adattípusait.

- Kapcsolatokból, amelyek leggyakrabban két, esetenként több egyedhalmazt kapcsolnak össze. Ha két egyedhalmaz között kapcsolat van, akkor ezt egy nyíllal jelöljük (a nyílra írva a kapcsolat megnevezését): a nyilat úgy értelmezzük (és eszerint alakítjuk ki a jelölést), hogy ha egy nyíl az A egyedhalmazból a B egyedhalmazba mutat, akkor minden A-beli egyedhez pontosan egy B-beli egyed tartozik (ekkor azt nem állíthatjuk, hogy minden B-beli egyedhez pontosan egy A-beli egyed tartozik). Kétirányú nyíl esetén az egyes egyedhalmazok egyedei kölcsönösen egyértelműen feleltethetők meg egymásnak. Ha egy kapcsolatnak kettőnél több egyedhalmaz résztvevője van, akkor az egyik egyedhalmazra mutató nyíl azt jelenti, hogy az illető egyedhalmaz egyedei a másik egyedhalmazok egyedeinek a függvényében alakulnak.

A gyakorlatban mind az egyedhalmazokat mind pedig a kapcsolatokat egy-egy adatbázis-táblában szokás tárolni. Az egyedhalmazoknál az egyedeket egyértelműen azonosító (azaz a többi hasonló, halmazbeli egyedtől megkülönböztető) attribútumokat az adott halmaz kulcs-attribútumainak nevezzük (vagy egyszerűen kulcsnak). A kapcsolatokat reprezentáló adatbázis-tábla a kapcsolat által összekapcsolt egyedhalmazok kulcs- attribútumaiból álló sorokat tartalmaz. Amennyiben a két összekapcsolt tábla sorai között kölcsönösen egyértelmű megfeleltetést találunk, akkor a kapcsolatleíró tábla minden halmaz minden egyedének a kulcsához tárolja a másik halmazból hozzá kapcsolódó egyed kulcsát. Amennyiben a megfeleltetés nem egy-egyértelmű, akkor a kapcsolat jellemzői között érdemes rögzíteni további olyan jellemzőket, amelyek az ugyanazon egyedek között létrehozott kapcsolatokat valamilyen módon megkülönböztetik. Ilyen esetben ezt - a megkülönböztető - információt is érdemes felvenni a kapcsolatot reprezentáló adatbázis-táblába, hogy annak sorai lehetőleg egyértelműek legyenek (ezzel kihasználhatunk egyes, az adatbázisba épített konzisztencia-vizsgálati mechanizmusokat a tárolt adataink épségének védelme érdekében). Ez az egyértelműség persze nem kötelező jellegű: alkalmazásterületenként (és adatbázis-szerkezetektől függően) változik az, hogy ezt meg kell-e valósítani vagy sem. Ennek kérdéseivel ezen könyv keretein belül nem foglalkozunk részletesebben.

A továbbiakban a Mini SQL adatbázis által támogatott SQL nyelvi elemeket ismertetjük. Számos lehetőség nincs meg a Mini SQL-ben (pl. nézettáblák, beágyazott lekérdezések), így ezeket a Mini SQL adatbázis-kezelővel együtt nincs módunk használni. Részletesebb SQL ismertetőt számos adatbázisokkal foglalkozó tankönyv is tartalmaz - itt csak egy rövid, tömör emlékeztetőt adunk a Mini SQL által bevezetett megszorításokkal.

Megemlítjük, hogy egyszerűbb Java-alapú adatbázis-kezelő programok írásához számos eszköz (varázsló) használható, melyek részletes ismertetésével itt nem foglalkozunk. Az érdeklődő Olvasónak megtekintheti például a Bulletproof Corporation cég JDesignerPro tervezőeszközét, mely a
`http://www.bulletproof.com/JDesignerPro/download.htm`

címről ingyenesen letölthető, és segítségével viszonylag kis ráfordítással rövid időn belül (akár csak minimális Java kódolással) kulcsrakész adatnyilvántartó Java alkalmazásokat fejleszthetünk.

## M.4.1. Adattáblák létrehozása

A legtöbb adatbázis-kezelő számos adattípust biztosít az attribútumok lehetséges értékhalmazának kijelölésére. Ezen adattípusok nagy része az SQL szabványban van rögzítve; itt most áttekintjük a MiniSQL adatbázis-kezelő által biztosított adattípusokat:

- `CHAR(hossz)` : legfeljebb a paraméterében megadott hosszúságú karakterlánc
- `TEXT(hossz)` : tetszőleges hosszú karakterlánc lehet, de hatékonyan csak a paraméterében megadott számú karaktert képes tárolni (nem szabványos SQL adattípus, így használata hordozható alkalmazásokban nem ajánlott)
- `INT` : egész szám
- `REAL` : valós szám
- `UINT` : előjel nélküli egész
- `DATE` : dátumot reprezentáló adattípus (Nap-Hó-Év formában, például: `12-5-1974`)
- `TIME` : időpontot reprezentáló adattípus (Óra:Perc:Másodperc formában, például: `23:12:59`)
- `MONEY` : numerikus érték, a tizedesvessző után 2 jegyet képes tárolni

Ezekkel a következő SQL tábladefiníciós utasítással tetszőleges táblák létrehozhatók:

```
CREATE TABLE táblanév (oszlopnév oszloptípus [ not null ]
                       {, oszlopnév oszloptípus [ not null ] }
```

Emlékezzünk, hogy a not null záradék az illető oszlopban tárolt értékeknél megtiltja az ismeretlen érték (NULL) tárolását.

## M.4.2. Adattáblák lekérdezése

Az adattáblák tartalmának lekérdezését a SELECT utasítással végezhetjük. Lehetőség van megadni, hogy mely oszlopokat akarjuk lekérdezni, milyen kritériumoknak megfelelő sorok érdekelnek minket, néhány egyéb elemmel. A SELECT utasítás általános alakja:

```
SELECT [táblanév.]oszlopnév { , [táblanév.]oszlopnév}
            FROM táblanév [ = táblaálnév] { , táblanév [ = táblaálnév] }
            [ WHERE [táblanév.] oszlopnév MŰVELETKÓD érték
            { AND | OR [táblanév.]oszlopnév MŰVELETKÓD érték } ]
            [ ORDER BY [táblanév.]oszlopnév [DESC] [,
            [táblanév.]oszlopnév [DESC] ]
```

A SELECT utasítás első sorában adjuk meg azon oszlopok neveit, amely oszlopokat a lekérdezésünk eredményeként vissza akarjuk kapni (ezeknek az oszlopoknak a tartalma érdekel minket). Ha az eredmény minden lehetséges oszlopa érdekel minket, ide írhatunk egy csillag karaktert. Oszlopnévként szükség esetén (például névütközéskor) használhatunk minősített oszlopneveket: az oszlopnév előtt egy ponttal elválasztva megadhatjuk,

hogy melyik tábla melyik oszlopára hivatkozunk (szükség esetén a táblanevek helyett egy álnevet is használhatunk, ahol a táblanevek-táblaálnevek egymáshoz rendelése a FROM záradékban történik).

A FROM záradékban kell felsorolni azon táblákat, amelyekből az eredményoszlopokat várjuk. Ha ugyanaz a tábla többször is előfordul a lekérdezésben (például ha a tábla sorait önmagukkal akarjuk összekapcsolni), álnevet adhatunk az egyes előfordulásoknak, megkönnyítve ezzel az egyes táblapéldányokra, illetve azok oszlopaira vonatkozó minősített hivatkozást. Megvalósításhoz tartozik ugyan, de érdemes megemlíteni, hogy az itt felsorolt táblák direktszorzataként előállt (óriás-)tábla sorai lesznek a hátrább megadott kritériumok szerint szűrve - a lekérdezés így kerül végrehajtásra.

A WHERE záradékban a minket érdeklő (kiválasztott) sorokra vonatkozó kiválasztási kritériumokat fogalmazhatunk meg. Például azt, hogy valamely tábla valamely oszlopának értéke nagyobb/kisebb egy előre megadott értéknél, esetleg több ilyen kritériumot összekapcsolhatunk logikai ÉS (AND) vagy pedig logikai VAGY (OR) művelettel. A műveletkód lehet <,>,<=,<>,>=,=, valamint használhatjuk a LIKE operátort is (ez mintaillesztésre használható; általános alakja: *s LIKE p* ahol s egy karakterlánc, p pedig egy minta - karakterlánc; az adatbázis-kezelő mintaillesztéssel keresi a megfelelő sorokat, ahol a minta egyes karaktereit egy az egyben illeszti az adatbázisban tárolt értékekre; a százalék jel a mintában a szöveg tetszőleges számú tetszőlegesen sok karakterére illeszkedik, míg az aláhúzás jel pontosan egy darab, de tetszőleges karakterre illeszkedik). Az érték mezőben vagy egy konstans értéket vagy pedig egy oszlopnevet adhatunk meg.

Végül az ORDER BY záradékban megadhatjuk, hogy az eredményt mely oszlopok szerint rendezve kívánjuk megkapni (ha több oszlopot adunk meg, akkor az adatbázis-kezelő egy hierarchikus lexikografikus rendezést végez el a kapott sorokon). Alapértelmezés szerint nagyság szerint növekvő rendezés történik, míg egy oszlop neve után a DESC kulcsszót megadva fordított irányú (csökkenő sorrendű) rendezést kapunk.

Megjegyezzük, hogy a Mini SQL alkalmazásokban a LIKE mellett lehetőség van CLIKE, RLIKE, SLIKE operátorok használatára is: az előbbi figyelmen kívül hagyja a kis- és nagybetűk közötti különbségeket; RLIKE esetén tetszőleges UNIX-ból ismert reguláris kifejezést használhatunk a minta illeszkedésének a vizsgálatához; SLIKE esetén a szó hangzásának a hasonlósága a döntő az illeszkedés eldöntésekor. Ezek a lehetőségek hiányoznak az SQL szabványból, így ha hordozható programokat akarunk készíteni, használatuk nem ajánlott. Amennyiben mintaillesztéskor a mintában speciális (százalék vagy aláhúzás) karakterre való illeszkedést kell vizsgálnunk, úgy ezen karakterek speciális jelentése eltakarható egy eléjük tett karakterrel. Megjegyezzük továbbá, hogy a WHERE és az ORDER BY záradék elhagyható. A WHERE elhagyása esetén az összes sor ki van választva.

Példa:

Az alábbi SQL utasítás a `Szemelyek` táblából kiválogatja a 65 évnél fiatalabb személyek nevét, címét és életkorát.

```
SELECT Nev, Cím, Eletkor FROM Szemelyek WHERE Eletkor < 65
```

## M.4.3. Sorok törlése

Adattáblák sorainak törlését a DELETE utasítással végezhetjük. Ennek általános alakja a következő:

```
DELETE FROM táblanév
                    WHERE oszlopnév MŰVELETKÓD érték
                    { AND | OR oszlopnév MŰVELETKÓD érték }
```

A fenti utasítás hatására a megadott táblából törölve lesznek mindazon sorok, amelyek megfelelnek a WHERE záradékban megadott feltételeknek (a lehetséges műveleteket és értékeket lásd a SELECT SQL utasítás rövid ismertetésénél).

Példa:

Az alábbi SQL utasítás törli a `Szemelyek` táblából azokat a sorokat (azon személyek adatait), akik 65 évnél fiatalabbak.

```
DELETE FROM Szemelyek WHERE Eletkor < 65
```

## M.4.4. Sorok beszúrása

Adattáblákba új sort az INSERT SQL utasítással szúrhatunk be. Ennek általános alakja a következő:

```
INSERT INTO táblanév [ ( oszlopnév { , oszlopnév } ) ]
                     VALUES ( érték { , érték } )
```

A fenti SQL utasítás az INTO után megadott nevű táblába beszúrja a VALUES mögött megadott értékeket, ahol az egyes értékek a táblanév mögött megadott sorrendben rendre az ott megadott oszlopokba kerülnek tárolásra.

Példa:

Az alábbi sor beszúrja egy új személy adatait a `Szemelyek` táblába.

```
INSERT INTO Szemelyek (Nev, Cím, Neme, Eletkor)
              VALUES ("Nagy Éva", "Profit u. 23", "N", 22)
```

## M.4.5. Sorok módosítása

Adattábla sorainak (bizonyos oszlopok tartalmának a) módosítására az UPDATE utasítást használhatjuk. Ennek általános alakja a következő:

```
UPDATE táblanév SET oszlopnév=érték { , oszlopnév=érték }
                    WHERE oszlopnév MŰVELETKÓD érték
                    { AND | OR oszlopnév MŰVELETKÓD érték }
```

A fenti utasítás a megadott nevű tábla kiválasztott sorait úgy módosítja, hogy az illető sorok megadott oszlopainak értékét az UPDATE utasításban megadott értékűre változtatja. A WHERE utáni záradékban határozzuk meg, hogy a tábla mely sorainak tartalmát akarjuk módosítani.

Példa:

Az alábbi SQL utasítás a `Szemelyek` tábla `Nagy Éva` nevű egyedének életkorát növeli meg eggyel.

```
UPDATE Szemelyek SET Eletkor=23 WHERE Nev = "Nagy Éva"
```

# M.5. Implementációs és tervezési feladat

Ebben a pontban felvázoljuk egy oktatási intézmény tanulmányi adatait tartalmazó adatbázisának szerkezetét. Az Olvasó gyakorlásképpen készítse el a hátterében álló adatmodellt, valamint az ezeket kezelő programokat. Ahol lehet és van értelme, az implementáció során biztosítsunk webes hozzáférést az adatbázishoz! A webes hozzáférés megvalósítható akár grafikusan egy Java appletben, akár karakteres (például lynx) webböngészőre

alapulva (ilyenkor a programlogikát a szerveroldalra helyezhetjük CGI-programok vagy servletek formájában).

A modell vázlata a következő:

Egy oktatási intézményben a képesítés megszerzéséhez bizonyos tantárgyakat kell elvégezni. Az elvégzendő tantárgyaknak van egy rákövetkezési lánca: megadva, hogy mely tantárgy felvételének mely másik tantárgyak elvégzése az előfeltétele. Az egyes tantárgyak egy vagy több szemeszter alatt végezhetőek el (ez a tantárgy egy jellemzője, illetve az egyes tantárgyakról tudni lehet, hogy a tárgy elvégzése mely szemeszterében heti hány órás lefoglaltságot jelent a tanulóknak). Feladatunk ez alapján egy ajánlott teljesítési terv (tantervi háló) kialakítása: ennek eredménye tantárgyaknak egy szemeszterekre bontott rákövetkezési listája, amely figyelembe veszi az egy szemeszter alatt elvégezhető tanórák számát (ez egy a felhasználó által megadott 16-32 óra hetenkénti leterhelést jelenthet), valamint figyelembe veszi a tanulók azon igényét, hogy a képesítést a fenti keretek között a lehető legkevesebb számú szemeszter elvégzése után lehessen megszerezni. Biztosítsunk lehetőséget egyéni tantervi hálók kialakítására is, ami akkor lesz különösen érdekes, amikor egy tárgy valamely részének elvégzése sikertelen, és a befejezéshez az illető tárgyból való bukás miatt a szokásos ütemterv módosítása szükséges (vagy egy-egy tárgy valamelyik szemeszterben nem kerül meghirdetésre, és ehhez is igazodni kell - ld. a következő pontot).

Az oktatási intézményben minden év két szemeszterből áll (egy nyári és egy téli szemeszterből). Minden szemeszterben meghirdetnek bizonyos kurzusokat. Ezt egy kurzusjegyzék-adatbázisban tárolják, amely évről-évre változhat (ezért az adatbázist szükséges lehet úgy felépíteni, hogy egy új év megkezdésekor ne kelljen annak tartalmát üríteni, hanem bővíthető legyen az azévi kínálattal). Egy kurzus egy tantárgy egy-egy szemeszterre eső tanegysége. Egy-egy kurzus az azt meghirdető oktatóval képez egy egyértelműen azonosítható tanegységet. Az egyes tantárgyak tanegységeit egyszerre több oktató is meghirdetheti. A tanulók mindegy melyik oktató által meghirdetett kurzuson végeznek el egy-egy, a tanulmányaikhoz szükséges tantárgyat. A lényeg az, hogy a tanulók a képesítés megszerzéséhez elvégzendő tantárgyak minden tanegységét elvégezzék. Az egyszerűség kedvéért tételezzük fel, hogy minden kurzus egy vizsgával végződik - nincsenek több tárgyat felölelő közös vizsgák, szigorlatok.

Szükséges lehet az oktatók nyilvántartása a kurzusok meghirdetéséhez (nyilvántartva például az oktató neve mellett elérhetőségi paraméterét, esetleg fogadóórájának helyét és időpontját más közérdekű adatokkal együtt). Az oktatóknak a nyilvános (és nem nyilvános) adatainak kezelésére képes programok mellett szükséges egy kurzusmeghirdetési program. Egy kurzus meghirdetésekor meg kell adni többek közt a kurzus tervezett időpontját, és a kurzusra jelentkező tanulók maximális számát (a programnak kell keresnie egy kellően nagy, abban az időpontban szabad tantermet a rendelkezésre álló tantermek között). Amennyiben egy kurzusra a maximálisan megengedettnél több hallgató is jelentkezik, úgy a meghirdetőnek lehetősége kell legyen kijelölni a kurzusra ténylegesen felvett hallgatókat (ehhez szükséges lehet a jelentkezési sorrendre, valamint olyan tényezők is befolyásolhatják, hogy ki vett részt az első kurzuson, stb.). Biztosítsunk egy, a vizsgákra kapott érdemjegyek bejegyzésére alkalmas programot is.

A tanulók nyilvántartása a tanulók személyi adatai mellett tárolják a tanuló tanulmányi előmenetelét: a már elvégzett tárgyakat (kurzusokat), az azokra kapott érdemjegyekkel. Tárolni kell továbbá a felvett, de még el nem végzett kurzusokat is (itt érdemjegy a szemeszter végén letett vizsgák után várható). A tanulók részére biztosítani kell egy kurzusokra jelentkezést lehetővé tevő programot. Ügyelni kell arra, hogy egy tanuló ne jelentkezhessen ugyanarra a tárgyra egyszerre több tanárnál (de egy bizonyos határidő előtt adjunk lehetőséget átjelentkezésre egyik kurzusról a másikra).

Végül szükség van a rendelkezésre álló tantermek adatbázisára, amely tárolja az illető tantermek jellemzőit (például férőhelyek száma, kiépítettség - pl. projektor, írásvetítő, képmagnó vagy hálózati csatlakozó megléte). Ezen túlmenően szükséges az egyes szemeszterekben a termek foglaltságának nyilvántartása is.

Végezetül biztosítsunk egy alkalmazást, amely képes ellenőrizni, hogy egy tanuló teljesítette-e az általa megszerezni kívánt képesítéshez szükséges összes tantárgyat.

A feladat elemzése után készítsük el az adatbázistervet, majd a szükséges alkalmazásokat. Megjegyezzük, hogy ez egy nagyobb lélegzetvételű feladat, amelynek részletes kidolgozására és megvalósítására akár egy-két hónapra is szükség lehet - ezzel nagyjából tartalmilag teljesítjük egy szakdolgozat követelményeit.