

## Adatbázisrendszerek II. – 6. Practice

**Topic:** JDBC alapú alkalmazásfejlesztés, Projekt 5

**Repository:** NEPTUNKOD\_DB2Gyak

**Folder:** NEPTUNKOD\_0320

**Forrás file-k:** Projekt feltöltése

**Töltse fel a GitHub rendszer aktuális mappába a jegyzőkönyvet és forrás fájlokat!**

**Határidő:** aktuális gyakorlat időpontja, ill. módosítás esetén 2025.03.26.

### Feladatok

**Projekt név:** NEPTUNKOD\_JDBC4

**Csomagnév:** neptunkoddb2

Adott a következő relációs szerkezet!



Az előző projekt folytatásaként készítsen **oldja meg a következő feladatokat:**

1. Kezelje le az idegenkulcs kapcsolatból eredő anomáliákat magyar nyelvű hibaüzenetekkel:
  - Hibás beszúrás esetén (kapcsolótáblába),
  - Hibás törlés esetén (mindkét alaptáblára),
  - Hibás módosítás esetén (kapcsolótáblában).

2. Készítsen működő menürendszert az egyik alaptáblához.

*Megoldandó feladatok:*

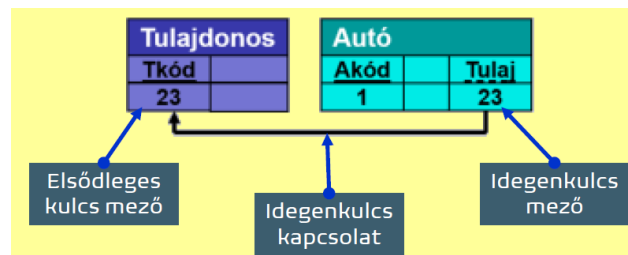
- Listázás,
- Beszúrás,
- Törlés,
- Módosítás,
- Kilépés a programból.

Adott a következő relációs modell!



## Anomáliák kezelése

- Két tábla között a megadott idegenkulcs azt jelenti, hogy az egyik táblában létrehozunk egy idegenkulcs mezőt (FOREIGN KEY ... REFERENCES ...), amit hozzákapcsolunk egy másik tábla elsődleges kulcsmezőjéhez.
- Innentől az idegenkulcs mezőben csak olyan érték lehet, ami a hozzákapcsolt tábla kulcsmezőjében megtalálható.
- Igaz ez a *beszúrás*, a *törlés* és a *módosítás* műveletére is.



## Anomáliák ellenőrzése összekapcsolt tábláknál



- A jelölt idegenkulcs azt jelenti, hogy az **Autó** táblában a **Tulaj** mező csak olyan értéket vehet fel, ami benne van a **Tkód** mezőben!
- 1. Tulajdonost tetszőlegesen felvihetünk (nem kell ellenőrizni).
- 2. Autó beszúrásakor csak olyan **Tulaj** értéket fogadjon el a program, ami létezik a **Tulajdonos** tábla **Tkód** mezőjében!
- 3. A Tulajdonos tábla **Tkód** mezőt ne engedje a program módosítani!
- 4. Az **Autó** táblában a **Tulaj** értékét csak olyanra engedje módosítani a program, ami **Tkód**-ként létezik a **Tulajdonos** táblában, de lehessen **NULL** értéket is megadni (ez a tulajdonos kódjának törlési lehetősége az **Autó** táblából)
- 5. Autót tetszőlegesen törölhetünk.
- 6. Olyan Tulajdonost nem törölhetünk, akihez tartozik autó!

## Anomáliák ellenőrzése összekapcsolt tábláknál



- Ugyanez a helyzet N:M kapcsolat esetén is:
  - A jelölt idegenkulcs azt jelenti, hogy az **Dolgozik** táblában az **ID** mező csak olyan értéket vehet fel, ami benne van a **Hallgató** tábla **ID** mezőben, a **Dolgozik** táblában a **Pkód** pedig olyat, ami megtalálható a **Projekt** tábla **Pkód** mezőben.
1. Olyan **Hallgatót**, akinek szerepel a **ID**-je a **Dolgozik** táblában, nem törölhetünk.
  2. Olyan **Projektet**, amelyiknek szerepel a **Pkódja** a **Dolgozik** táblában, nem törölhetünk.
  3. A **Dolgozik** táblából tetszőlegesen törölhetünk sorokat.

## Anomália kezelés



### A táblák tartalma:

Hallgató					Dolgozik		Projekt				
<u>ID</u>	Vnév	Knév	Szüldő	Lakcím	<u>ID</u>	<u>Pkód</u>	<u>Pkód</u>	Név	Összeérték	Indul	Zárul
1	Kis	Éva	2000.03.19	Eger C.u.1	1	1	1	Ragacs	25000000	2022.03.01	2024.03.01
2	Nagy	Joe	1985.07.11	Miskolc A.u 4.	1	2	2	Golyó	51234500	2020.07.01	2025.12.31
3	Kép	Béla	1998.12.07	Ózd F.u.13.	2	1	3	Lufi	10000000	2024.04.01	2025.12.31
					2	3					
					3	1					

### Próbáljuk beszúrni egy hibás rekordot:

```
String sqlp = "Insert Into Dolgozik Values(4, 4)";
dbm.CommandExec(sqlp);
```

COMMAND: Insert Into Dolgozik Values(4, 4)  
Command OK!

Ennek nem lett volna szabad megtörténnie! Oka?

## Anomália kezelés

Mi az oka annak, hogy be tudunk szűrni hibás idegenkulcsot?

Az, hogy az idegenkulcsok figyelése alapból ki van kapcsolva az SQLite-ban. Be kell kapcsolnunk minden művelet előtt!

A legegyszerűbb megoldás, ha átírjuk úgy a CommandExec metódust, hogy a parancs kiadása előtt kapcsolja be az idegenkulcs figyelését:

```
public void CommandExec(String command) {  
    Connection conn= Connect();  
    String sqlpP = "PRAGMA foreign_keys=on;";  
    String sqlp = command;  
    SM("Command: "+sqlp);  
    try {  
        Statement s = conn.createStatement();  
        s.execute(sqlpP);  
        s.execute(sqlp);  
        SM("Command OK!");  
    } catch (SQLException e) {  
        SM("CommandExec: "+e.getMessage());  
    }  
    Disconnect(conn);  
}
```

## Anomália kezelés - INSERT

Töröljük le a hibás rekordot az Autó táblából:

```
Delete from Dolgozik where id = 4
```

A kód a Program osztályban:

```
String sqlp = "Delete from Dolgozik where id = 4";  
dbm.CommandExec(sqlp);
```

■ Próbáljuk újból beszúrni a hibás rekordot:

```
String sqlp = "Insert Into Dolgozik Values(4, 4)";  
dbm.CommandExec(sqlp);
```

```
COMMAND: Insert Into Dolgozik Values(4, 4)  
CommandExec: [SQLITE_CONSTRAINT] Abort due to constraint violation  
(FOREIGN KEY constraint failed)
```

## Anomália kezelés

### ■ A CommandExec metódus megtanítása magyarul:

```
public void CommandExec(String command) {  
    Connection conn= Connect();  
    String sqlp = "PRAGMA foreign_keys=on;";  
    String sql = command;  
    SM("Command: "+sql);  
    try {  
        Statement s = conn.createStatement();  
        s.execute(sqlp);  
        s.execute(sql);  
        SM("Command OK!");  
    } catch (SQLException e) {  
        String msg = e.getMessage();  
        if (msg.contains("FOREIGN KEY constraint failed")) msg = "Hibás Idegenkulcs érték!";  
        SM("CommandExec: "+msg);  
    }  
    Disconnect(conn);  
}
```

COMMAND: Insert Into Dolgozik Values(4, 4)  
CommandExec: Hibás Idegenkulcs érték!

## Anomália kezelés – delete

### ■ A táblák tartalma:

Hallgató					Dolgozik		Projekt				
ID	Vnév	Knév	Szüldő	Lakcím	ID	Pkód	Pkód	Név	Összeérték	Indul	Zárul
1	Kis	Éva	2000.03.19	Eger C.u.1	1	1	1	Ragacs	25000000	2022.03.01	2024.03.01
2	Nagy	Joe	1985.07.11	Miskolc A.u 4.	1	2	2	Golyó	51234500	2020.07.01	2025.12.31
3	Kép	Béla	1998.12.07	Ózd F.u.13.	2	1	3	Lufi	10000000	2024.04.01	2025.12.31
					2	3					
					3	1					

### ■ Próbáljuk törölni az 1-es ID-jű hallgatót:

```
String sqlp = "Delete from hallgató where id = 1";  
dbm.CommandExec(sqlp);
```

COMMAND: Delete from hallgató where id = 1  
CommandExec: Hibás Idegenkulcs érték!

Mivel a Dolgozó táblában van hivatkozás erre a rekordra, az SQLite szintén a **FOREIGN KEY constraint failed** hibaüzenetet adja, és ezért jelenik meg a magyar nyelvű üzenet!

## Anomália kezelés - Update

### ■ A táblák tartalma:

Hallgató					Dolgozik		Projekt				
ID	Vnév	Knév	Szüldő	Lakcím	ID	Pkód	Pkód	Név	Összeérték	Indul	Zárul
1	Kis	Éva	2000.03.19	Eger C.u.1	1	1	1	Ragacs	25000000	2022.03.01	2024.03.01
2	Nagy	Joe	1985.07.11	Miskolc A.u. 4.	1	2	2	Golyó	51234500	2020.07.01	2025.12.31
3	Kép	Béla	1998.12.07	Ózd F.u.13.	2	1	3	Lufi	10000000	2024.04.01	2025.12.31
					2	3					
					3	1					

### ■ Próbáljuk (hibásan) módosítani a dolgozó tábla 1. sorát:

```
String sqlp = "Update dolgozik set id = 4 where id = 1 and pkód = 1";  
dbm.CommandExec(sqlp);
```

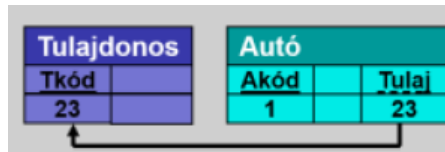
**COMMAND:** Update dolgozik set id = 4 where id = 1 and pkód = 1

**CommandExec:** Hibás Idegenkulcs érték!

Nincs túl sok hibaüzenet az SQLite-ban, ezért viszonylag egyszerű magyarázni a programot. :-)

## Anomália kezelés v2

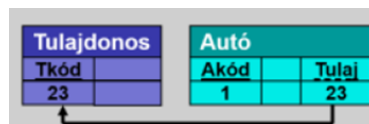
Egy **másik megoldás**: nem a parancs kiadását, hanem a *parancshoz beolvasott értékeket ellenőrizzük*:



Írjon egy olyan metódust, amivel egy paraméterrel megadható nevű táblában, egy paraméterrel megadható feltételnek megfelelő *rekord darabszámát tudjuk meghatározni*:

```
public int SelectCount(String table, String condition) {
    int pc=-1;
    String sqlp= "SELECT count(*) From "+table+" where "+condition;
    SM(sqlp);
    Connection conn= Connect();
    try {
        Statement s = conn.createStatement();
        ResultSet rs = s.executeQuery(sqlp);
        while(rs.next()) {
            pc = rs.getInt(1);
        }
        rs.close();
    } catch (SQLException e) {SM(e.getMessage());}
    Disconnect(conn);
    return pc;
}
```

## Anomália kezelés v2



Írja meg annak a **Programnak** a kódját, amivel egy beolvasott **Tkod** létezését tudjuk ellenőrizni a **Tulajdonos** táblában:

```
dbm.SM("Tulajdonos darabszáma Tkod alapján ");
String tkod = cm.ReadData("Kérem a Tkod értékét ");
int db = dbm.SelectCount("Tulajdonos", "Tkod = "+tkod);
dbm.SM("Eredmény: "+db);
```

```
Tulajdonos darabszáma Tkod alapján
Kérem a Tkod értékét
2
SELECT count(*) From Tulajdonos where Tkod = 2
Eredmény: 1
```

```
Tulajdonos darabszáma Tkod alapján
Kérem a Tkod értékét
8
SELECT count(*) From Tulajdonos where Tkod = 5
Eredmény: 0
```

Ha jól működik,  
vegyük ki az  
összeállított  
parancsot  
képernyőre író  
sort a  
SelectCount  
metódusból.

## Anomália kezelés v2

Írja meg annak a **Programnak** a kódját, amivel egy **Tulaj** értéket tudunk módosítani az **Auto** táblában:

Tulajdonos		Autó	
Tkód		Akód	Tulaj
23		1	23

```
dbm.SM("Tulajdonos kódjának módosítása az Autó táblában ");
String rsz = cm.ReadData("Kérem az Autó rendszámát: ");
int db = dbm.SelectCount("Auto", "Rendszam = '"+rsz+"'");
if (db==0) dbm.SM("Hibás rendszám!");
else {
    String tkod = cm.ReadData("Kérem a Tulajdonos kódjának az értékét: ");
    db = dbm.SelectCount("Tulajdonos", "Tkod = '"+tkod+"'");
    if (db==0) dbm.SM("Ehhez a kódhoz nem tartozik tulajdonos, így nem adható meg!");
    else {
        String sqlp= "UPDATE Auto SET Tulaj = '"+tkod+" WHERE Rendszam = '"+rsz+"'";
        dbm.CommandExec(sqlp);
    }
}
```

- 1 Létező rendszám ellenőrzése
- 2 Létező tulajdonos ellenőrzése
- 3 Megfelelő paraméterek esetén a Tulaj érték módosítása az Auto táblában

## Anomália kezelés v2

A Program futtatása:

Tulajdonos kódjának módosítása az Autó táblában

Kérem az Autó rendszámát:

123456

Hibás rendszám!

Tulajdonos kódjának módosítása az Autó táblában

Kérem az Autó rendszámát:

CHR411

Kérem a Tulajdonos kódjának az értékét:

5

Ehhez a kódhoz nem tartozik tulajdonos, így nem adható meg!

Tulajdonos kódjának módosítása az Autó táblában

Kérem az Autó rendszámát:

CHR411

Kérem a Tulajdonos kódjának az értékét:

3

Command: UPDATE Auto SET Tulaj = 3 WHERE Rendszam = 'CHR411'

Command OK!

## Anomália kezelés v2

Célszerű a megírt kódot beírni egy **DbMethods** osztályban létrehozott metódusba, és onnan a megfelelő menü kiválasztásakor elindítani.



A *SelectCount* és a *CommandExec* metódusok felhasználásával az összes anomália ellenőrizhető, kezelhető.

Természetesen ez csak egy lehetőség a sok közül, más megoldásokkal is lehet kezelni az anomáliákat.

## Menürendszer

Lényegesen elegánsabb egy program, ha nem kell a főprogramban az egyes funkciók eléréséhez a kódot átírni, hanem egy menürendszerrel vezérelhető.

### Készítsen egy menürendszert:

– A példakódban 4 menüpont szerepel.

– Ha a félèves feladatot konzolos technológiával oldja meg, sokkal több menüre lesz szüksége, sőt, javaslom, hogy többszintű menürendszert találjon ki, pl:

- Első szinten kiválasztja a kezelendő táblát.
- A második szinten a táblához tartozó elvégzendő műveletet.

Amikor beolvassuk a kiválasztott menü értékét, ellenőriznünk kell, hogy:

- Van-e beírva adat, vagy üres string a beolvasott érték.
- A beolvasott érték számmá alakítható-e.
- Ha számmá alakítható, az értéke 0-3 közötti-e.
- Ezeket ellenőrzi a **test(s)** metódus!

```
Menü
=====
0. Kilépés
1. Listázás
2. Beszúrás
3. Törlés
Add meg a választott menü számát:
```

## Menürendszer

A menü metódus a **Programban**:

```
static void menu() {
    SM("\n");
    SM("Menü");
    SM("=====");
    SM("0. Kilépés ");
    SM("1. Listázás ");
    SM("2. Beszúrás ");
    SM("3. Törlés ");
    String ms = cm.ReadData("Add meg a választott menü számát: ");
    int m = -1;
    if (test(ms)) m = StringToInt(ms);
    switch(m) {
        case 0: SM("A program leállt! "); System.exit(0); break;
        case 1: SM("Listázás végrehajtása"); break;
        case 2: SM("Beszúrás végrehajtása"); break;
        case 3: SM("Törlés végrehajtása");
    }
}
```

Az egyes funkciókat metódusokba kell szervezni, és a metódusokat kell meghívni a case szerkezetben (a kiírások helyett).

## Menürendszer

A **test** metódus a Programban:

```
static boolean test(String s) {  
    if (s.length() == 0) {  
        SM("Próbáld újra!");  
        return false;  
    }  
    else  
        try {  
            int x=Integer.valueOf(s);  
            if (x>=0 && x<4) return true;  
            else {  
                SM("Mintha nem lenne 0-3 közötti a szám! Próbáld újra!");  
                return false;  
            }  
        } catch (NumberFormatException nfe) {  
            SM("Ez nem tűnt jó adatnak! Próbáld újra!");  
            return false;  
        }  
}
```

## Menürendszer

A **StringToInt** metódus a Programban

---

```
static int StringToInt(String s) {  
    int x = 0;  
    try {  
        x=Integer.valueOf(s);  
    } catch (NumberFormatException nfe) {}  
    return x;  
}
```

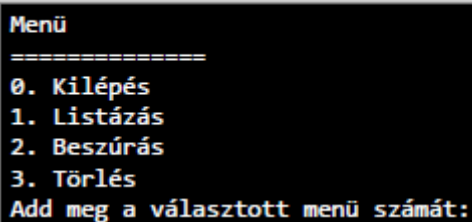
---

## Menürendszer

A menürendszer futtatása a Programban.

```
while (1!=0){  
    menu();  
}
```

A futási képernyő:



```
Menü  
=====  
0. Kilépés  
1. Listázás  
2. Beszúrás  
3. Törlés  
Add meg a választott menü számát:
```