Mérési jegyzőkönyv – Adatbázisok Laboratórium

III. mérés: JDBC

Név:	**Szabó Bence Farkas**
Neptun kód:	**RF57V5**
Feladat kódja:	**30 - VASUT**
Mérésvezető neve:	Csapó Tamás
Mérés időpontja:	**2018-03-19 12:15**
Mérés helyszíne:	**HSZK N**
A működő alkalmazás elérhetősége:	http://rapid.eik.bme.hu/~rf57v5/jdbc/
Felhasználónév:	rf57v5
Jelszó:	bence1995
Megoldott feladatok:	**1,2,3,4.1**
Elérhető pontszám (plusz pontok nélkül):	**40p**

Felhasználói útmutató

Az alkalmazás egy adatbázis kezelését teszi lehetővé grafikus felülettel. Az adatbázis egy vasúttársaság adatait tárolja, három tábla segítségével. Van egy tábla, melyben a járatok adatait tároljuk (vonatszám, típus, mely napokon közlekedik, mettől- és meddig közlekedett, megjegyzés), egy melyben az állomások adatait (állomás id, állomás neve, város neve), illetve egy megáll tábla melyben tároljuk, hogy egy járat megáll egy állomáson (id, vonatszám, állomás id, érkezés, indulás).

A programmal megtekinthetjük a járatokat, új járatot vehetünk fel az adatbázisba, hozzárendelhetünk egy új járatot egy állomáshoz a megáll táblában.

Az alkalmazás a http://rapid.eik.bme.hu/~rf57v5/jdbc/ címen érhető el, melyben az rf57v5 felhasználónévvel és bence1995 jelszóval kapcsolódhatunk az adatbázishoz.

Mérési feladatok megoldása

1. feladat:

A megoldáshoz használt SQL utasítás

```
SELECT vonatszam, tipus, megjegyzes FROM jarat

SELECT vonatszam, tipus, megjegyzes FROM jarat WHERE vonatszam LIKE ?
ESCAPE '@'
```

Magyarázat

A feladat során a Controller searchEventHandler() és a VasutDal search(String keyword) metódusait kellett megvalósítani, illetve kicsit átalakítani a View.fxml-ben a searchTable oszlopait.

Elsőként a search függvénnyel kezdtem, melyben az adatbázishoz kapcsolódva, SQL lekérdezéssel kellett kinyerni az adatokat. Attól függően, hogy a paraméterként kapott keyword keresési feltétel üres e vagy sem, két SQL lekérdezést használtam, melyek fentebb láthatók.
Ha a keyword üres, akkor egyszerűen kilistázzuk az összes járat adatait, ha azonban valamilyen értéket tartalmaz, paraméteres lekérdezést használunk, melyben a keresési feltétel a keyword lesz. Ahhoz hogy a speciális karaktereket is feldolgozzuk, a lekérdezésben az ESCAPE '@' használtam. Az eredményként létrehozott listát visszatérési értékként adjuk át.

Következik a searchEventHandler() függvény, melyben beolvassuk a keresőmezőbe írt értéket és ezt átadva a search függvénynek (keyword paraméterként) lekérdezzük a járatok listáját. Ha a keresőmező üres volt, simán meghívjuk a search metódust. Ha azonban tartalmaz valamilyen értéket, konkatenáljuk elölről és hátulról is „%" jellel, hogy ne csak teljes illeszkedésre, hanem szó eleji, közepi és végi illeszkedésre is találatot kapjunk, majd ezek után meghívjuk a search metódust a már új keresési feltételünkkel.

A feladat megvalósításához még módosítani kellett a View.fxml-ben a searchTable oszlopait, illetve a Controller initialize() függvényében ezeket fel kellett venni, hogy a searchEventHandler-ben feltölthessük őket adatokkal.

2. feladat

A megoldáshoz használt SQL utasítás

```
SELECT vonatszam FROM jarat WHERE vonatszam LIKE ?

INSERT INTO jarat (vonatszam, tipus, nap, kezd, vege, megjegyzes)
VALUES (?, ?, ?, ?, ?, ?)

UPDATE jarat SET tipus=?, nap=?, kezd=?, vege=?, megjegyzes=?
WHERE vonatszam LIKE ?
```

Magyarázat

A feladat megoldásához a Controller editEventHandler() és a VasutDal insertOrUpdate() metódusait kellett megvalósítani, illetve a View.fxml-ben felvenni az új elemeket.

Először a View.fxml-ben felvettem a szükséges mezőket az editTab-ba, majd inicializátam őket a Controller osztályban.

Következett az editEventHandler, ahol létrehozunk egy Jarat objektumot, makd a beviteli mezőkből kiolvasott értékekkel feltöltjük azt adatokkal. Ez után átadjuk a Jarat objektumot a VasutDal insertOrUpdate függvényének ahol megtörténik a beszúrás vagy módosítás. A művelet eredményéről ActionResult-ban értesülünk melynek értékét kiírjuk a felhasználónak.

Az insertOrUpdate függvényben történik az adatmanipuláció lényegi része. Itt a fenti első SQL lekérdezéssel megnézzük, létezik e az adatbáziban az adott vontszámú járat. Ennek eredménye dönt arról, hogy insert vagy update műveletet kell végrehajtanunk. Ha az eredménytábla (ResultSet) üres, a járat nincs az adatbázisban, tehát insert műveletre van szükség. Ehhez a fent látható második, paraméteres SQL utasítást használtam, melynek a PreparedStatement setXXX() függvényeivel beállítottam az értékeit a kapott Jarat objektum alapján. Ezek után végrehajtjuk a lekérdezést, majd ActionResult.InsertOccured-al térünk vissza, ha az sikeres. Sikertelen művelet esetén Exception keletkezik, melyet kezelve ActionResult.ErrorOccured-al térünk vissza.
Ha az eredménytábla nem üres, a járat már szerepel az adatbázisban, tehát update műveletre van szükség. Ehhez a fent látható harmadik SQL lekérdezést használom, melynek az insert-hez hasonlóan adjuk meg a paraméterek értékeit, majd futtatjuk azt. A sikeres eredményről szintén ActionResult.InsertOccured –al értesítjük a felhasználót, hiba esetén pedig az inserthez hasonlóan ActionResult.ErrorOccured-al.

Mintaadatok a teszteléshez:

Az első sor még nem létezik az adatbázisban, tehát beszúrás, a második sor adatait használva frissítés fog történni.

Vonatszam	Tipus	Nap	Kezd	Vege	Megjegyzes
100	Gyors	1110011	2000-01-01	2018-12-31	Beszúrás
213	Szemely	1111111	2012-01-01	2020-12-31	Frissítés

3. feladat

Magyarázat

A feladat megoldásához a Jarat osztály parse() metódusait kellett megvalósítani, módosítani az adatok feltöltését az editEventHandel() függvényben, illetve néhány apró módosítás kellett az insertOrUpdate függvényben.
Elsőnek a parse() függvényekkel kezdtem. Ahol nem kellett az adatoknak külön feltételnek megfelelniük, ott elég volt meghívni az adott mezű set() metódusát (pl.: megjegyzes, tipus). A többi mező formátumát regex kifejezésekkel ellenőriztem:

- Vonatszám – minimum 1 de maximum 5 jegyű számot várunk, ehhez a `[0-9]\\d{0,4}` kifejezést használtam, mely 0-9 karaktereket vár
- Nap – 7 karakter hosszú, 0-1 karakterekből álló számot várunk. Ehhez a `[0-1]\\d{6}` kifejezést használtam
- Kezd és Vege – Ide egy 'YYYY-HH-DD' formátumú dátumot várunk. Ezt a `[1-9]\\d{3}\\-[0-1]\\d\\-[0-3]\\d` kifejezéssel ellenőrizzük

Mivel az adatbázisban megengedett hogy a két dátum null értéket vegyen fel, ezért ezeket is kezelnünk kell. Ha a kapott **String** üres, null értéket adunk a dátumnak, amelyet később az **insertOrUpdate()** függvényben kezelünk.

A parse függvények hibás értékek esetén **ValidationException**-t dobnak, melyet az **editEventHandler**-ben kapunk el, és ezekről tájékoztatjuk a felhasználót.

A dátumok lehetséges null értékeit az **insertOrUpdate()** metódusban kezeljük, ahol ha a kapott dátum értéke null, a **PreparedStatement** paraméterének null értéket adunk át.

Az editEventHandler()-ben a Jarat objektum feltöltését kiegészítjük az új parse metódusokkal, így biztosan helyes értékek kerülnek bele. Ha a feltöltés közben ValidationException-t kapunk, kezeljük azt, és kiírjuk a felhasználó számára a hiba okát, hogy változtathasson a helytelen értékeken.

4. feladat:

A megoldáshoz használt SQL utasítás

```
SELECT id FROM allomas WHERE id LIKE ?


SELECT MAX(id) as value FROM megall


INSERT INTO megall (id, vonatszam, allomas_id, erk, ind)
VALUES (?, ?, ?, ?, ?)
```

Magyarázat

A feladat megoldásához implementálni kellett a **Controller commitEventHandler()**, a **VasutDal commit()**, **setAutocommit()** és **rollback()** metódusát, módosítani az **insertOrUpdate()** metódust, illetve felvenni néhány elemet a **View.fxml**-be.

Kezdetnek a felvettem egy új beviteli mezőt a **View.fxml**-ben amelybe az állomás azonosítóját íratjuk, illetve engedélyeztem a **Commit** gombot.

Ezek után következtek a **VasutDal** metódusai. A **setAutoCommit** metódusban a kért értékre állítjuk be az autocommit-ot az adatbázisban.

A **commit()** metódusban kiadjuk a commit parancsot (**connection.commit()**), amely ha sikeresen lefut, igaz értékkel tér vissza, ha exception keletkezik commit során, hamis értékkel térünk vissza.

Ehhez hasonló a **rollback()** metódus, mely siker esetén igaz, hiba esetén hamis értékkel tér vissza.

Az insertOrUpdate() metódusban paraméterként kapunk egy int értéket, ez lesz az idegen kulcs a lekérdezésben. A fent látható első SQL utasítással meggyőződünk arról hogy az adott ID-jú állomás létezik e az adatbázisban, ha nem ActionResult.ErrorOccured értékkel térünk vissza.

Ha az eredménytábla nem üres, haladunk tovább a már említett módon (vonatszám ellenőrzése, vonat beszúrása, stb.). Ezek után a fenti második SQL utasítással megnézzük, melyik a legnagyobb ID a MEGALL táblában. A kapott értéket egyel megnövelve felhasználjuk majd az új megálló felvételéhez (így biztosan egyedi lesz az ID). Ezek után az új ID-val, a kapott állomás ID-val és vonatszámmal létrehozunk egy megállót az adatbázisban (harmadik, paraméteres SQL utasítás). Az ERK és IND attribútumokat fix értékekkel töltjük fel. Ezek után visszatérünk ActionResult.InsertOccured-el.

Ezt az editEventHandler-ben kezeljük, melynek elején meghívjuk a setAutoCommit(false) függvényt, így a két insert művelet csak a commit meghívása után fut le (egy tranzakcióban). Itt dolgozzuk fel az állomás ID-jét amelyet átadunk az insertOrUpdate() függvénynek. Ennek a formátumát is ellenőriznünk kell, ehhez létrehoztam egy parseAllomas() nevű függvényt, mely a Jarat parse függvényeihez hasonlóan ellenőrzi a kapott String értékét.

A commit paracsot a commitEventHandler()-ben adjuk ki, amely a commit gomb eseménykezelője. Itt meghívjuk a VasutDal commit() metódusát, amely ha sikeresen lefut (igazzal tér vissza) értesítjük a felhasználót. Ha hiba keletkezik, meghívjuk a VasutDal rollback() függvényét és értesítjük róla a felhasználót.

Mintaadatok a feladathoz:

Az első sor sikeresen lefut, a második hibával leáll, mert nincs ilyen állomás az adatbázisban

Vonatszam	Tipus	Nap	Kezd	Vege	Megjegyzes	Allomas
101	Gyors	1110011			Sikeres	24
102	Szemely	1111111			Sikertele	500