Mérési jegyzőkönyv – Szoftver Laboratórium 5

3. mérés: JDBC

Név:	**Dávid Ádám**
Neptun kód:	**AO4S5C**
Feladat kódja:	**31-Legi1**
Mérésvezető neve:	**Nagy Tamás**
Mérés időpontja:	**2016-03-31 12:00**
Mérés helyszíne:	**HSZK B**
A működő alkalmazás elérhetősége:	**http://rapid.eik.bme.hu/~ao4s5c/jdbc/**
Megoldott feladatok:	**1,2,3,4.1,4.2,5**
Elérhető pontszám (plusz pontok nélkül):	**50p**

Felhasználói útmutató

Felhasználónév:ao4s5c

Jelszó: GFC3R

Amikor elindul az alkalmazás akkor be kell jelentkeznünk. Ezt úgy tehetjük meg, hogy beírjuk a felhasználónevet és a jelszót, majd a connect gombra nyomunk. Az alkalmazás 4 fület tartalmaz. Ebből az első a log, melyben a felhasználók fontos információkat kapnak egyes műveleteik sikerességeiről, illetve ha nem akkor hogy miért voltak sikertelenek. A második fül a Search fül. Itt beviteli mezőbe beírhatjuk annak a városnak a 3 betűs rövidítését melyről tudni szeretnék, hogy tartozik-e hozzá olyan járat, amely innen indul vagy ide érkezik. Ha megnyomjuk a Search gombot akkor ha vannak ilyen járatok akkor azoknak kilistázza a számat, indulási idejét és érkezési idejét. A List all gomb az összes járatot kilistázza.

A 3. fül az Edit fül. Ezen a fülön gépeket vehetünk fel. A csillaggal megjelölt mezők megadása kötelező, továbbá a dátumok formátumát odaírtuk azokhoz a mezőkhöz, amelyekhez dátumot várunk el. Ha a felhasználó úgy gondolja, hogy helyesen beírta az adatokat az Insert/Update gombra kell nyomjon. Amennyiben valami probléma adódik az adatok felvételénél azt a Log fülön tekinthetjük meg. Itt kiírja azt, hogy melyik mező formátumánál volt probléma. Ezután a felhasználó visszamehet az Edit fülre és kijavíthatja a mezők hibáit. Ha az Insert/Update gomb hatására már nincs hiba akkor az adatok bekerülhetnek az adatbázisba. Azonban ahhoz hogy az adatok valóban bekerüljenek az adatbázisba sikeres sor felvétele vagy frissítése után a Commit gombra kell nyomni.

A 4.fül egy statisztika mely ha rányomunk a statics gombra kiírja hogy mennyi a napi átlagos járatszám PRG-től HEL célállomásig.

Az alkalmazásból a jobb felső sarokban az X-el lehet kilépni

Mérési feladatok megoldása

1. feladat

A megoldáshoz használt SQL utasítás

Ha a feladat megoldásához szükség volt SQL utasításra, azt itt tüntessük fel, a Kód stílus segítségével!

```
SELECT szam, ind, erk FROM jarat;

SELECT szam, ind, erk FROM jarat
WHERE honnan LIKE ? OR hova LIKE ?
```

Magyarázat

Először készítettem egy új gombot melynek a felirata a „List all” felirat lett. Ennek a gombnak az eseménykezelője meghívja a controllerben lévő listAll(List<String> log) függvényt ez pedig meghívja a model-ben lévő listAll() függvényt. Ez a függvény az Első SQL utasítást használva visszaad egy olyan ResultSet-et mely tartalmazza a jarat minden sorát. Ha eközben bármi exception keletkezett, azt belerakjuk a logba és a viewban kiirjuk a log tartalmát.

A feladat másik fele hogy kilistázzuk azokat a járatokat, melynek indulási vagy érkezési helye megegyezik a kulcsszóval vagy a részletével. Ezt PreparedStatment-el oldottam meg. Ez alkalmas volt arra, hogy az SQL kódomba belerakjak különféle paramétereket, melyeket csak a kulcs ismerete után állítok be. Ezek az SQL kódban a „?” karakterek. A speciális karakterek kereshetőségét úgy oldottam meg, hogy ezeket a karaktereket még azelőtt megváltoztattam a kulcsszóban mielőtt azt beillesztettem volna az SQL kódba. Ehhez a kereséshez a „Search” feliratú gombot lehet használni. A log fájlba természetesen itt is minden hiba bekerül.

2. feladat

A megoldáshoz használt SQL utasítás

```
SELECT kod FROM gep
WHERE kod = ?

INSERT INTO gep (kod, tipus, musz_ell, javitas,
legcsavaros, ules, hatotav, megjegyzes)
VALUES(?, ?, ?, ?, ?, ?, ?, ?)

sqlUpdate = UPDATE gep
"SET tipus = ?, musz_ell = ?, javitas = ?, legcsavaros = ?,
ules = ?, hatotav = ?, megjegyzes =? WHERE kod = ?
```

Magyarázat

A feladat megoldásához a modelben „Insert/Update” feliratú gomb lenyomása esetén a hozzá tartozó eseménykezelő összegyűjti a TextFieldekből egy Map tárolóba a géphez tartozó különféle adatok, majd meghívja a controllerben lévő modifyData(Map data,boolean AutoCommit, List<String> log) függvényt mely ezután meghívja a modelben lévő modifyData(Map data, boolean AutoCommit) függvényt. Ebben a függvényben először az első

SQL utasítás segítségével eldöntöm, hogy az adott kódú gép már létezik-e. Amennyibe létezik, akkor frissítem a megfelelő sort a második SQL utasítás segítségével, azonban ha nem akkor új rekordot hozok létre az első SQL utasítást használva. Mind a kettő esetben egy PreparedStatmentet használok. A különböző paramétereket a Map típusú datából veszem ki, melyben a kulcsok az gep oszlopneveivel egyezik meg például:"KOD" vagy „MUSZ_ELL". Ha a függvény lefutása közben hiba lépett fel, akkor a hibát rögzítjük, és Error-ral térünk vissza. Sikeres végrehajtás esetén UpdateOccured-del vagy InsertOccured-del tér vissza a függvény, melyekből az előbbi a sikeres sorfrissítést jelenti, az utóbbi pedig a sikeres sor hozzáadását. A függvények lefutása után értesítjük a felhasználót az eredményről a Log-ban.

Példa Adatok:

KÓD	TÍPUS	MUSZ_ELL	JAVITAS	LEGCSAVAROS	ULES	HATOTAV	Megjegyzés
223	B445B	2014-04-05	2013-03-05	1	420	4000	Test1
11	B445B	2014-04-08	2014-03-02	0	300	18000	Test2
224	B4467	2014/05/05	2014-01-06	0	330	17000	
225	B4467	2014-05-07	2014-02-03			13200	DefaultTest
226B		2014-05-04	2011-01-09	1	420	22000	
220	B312		2011-07-08	0	890	12	

3. feladat

Magyarázat

A 3. feladatot úgy oldottam meg, hogy különböző formátumok mint a csak számokból álló bemenet vagy a dátumok ellenőrzésére különféle reguláris kifejezéseket írtam. Ehez a verifyData(Map data, List<String> log) metódust készítettem el. Ebben a metódusban készítettem String tömböket. Az egyik String tömben azokat a kulcsneveket tároltam melyekhez szükség van formátum ellenőrzéshez, egy másik String tömbben a felhasználók tájokoztatására raktam bele hibaüzeneteket, melyekből könnyen rájöhetnek, hogy melyik mezőbe írtak helytelen adatot. Van egy harmadik string tömb is ebben tárolom a reguláris kifejezéseket.

`([0-9]{1,7})`	KOD (1-7 db numerikus számú)
`[0-9a-zA-Z]{1,7}`	TÍPUS (bármilyen karakterből 1-7 db)
`[0-9]{4}-[0-1][0-9]-[0-3][0-9]`	MUSZ_ELL (A dátum formátuma ÉÉÉÉ-HH-DD)
`[0-9]{4}-[0-1][0-9]-[0-3][0-9]"`	JAVITAS (A dátum formátuma ÉÉÉÉ-HH-DD)
`[0-9]+`	ULES (Nem nulla darab szám)

`[0-9]+`	HATOTAV(Nem nulla darab szám)

Egy While() ciklussal végigmegyek ezeken a tömbökön és megvizsgálom minden adatra, hogy illeszkednek-e a rájuk vonatkozó reguláris kifejezésekre. Amennyiben nem illeszkedik valamelyik adat azt a log-ba rögzítem és a függvény false-al tér vissza. Ezt az értéket felhasználva döntöm el, hogy a második feladatban megírt függvényt egyáltalán meghívom-e.

4 Feladat

4.1 feladat

A megoldáshoz használt SQL utasítás

```
INSERT INTO menetrend (ID, kod, szam, nap)
VALUES(sorszam.nextval, ?, ?, 1)
```

Magyarázat

A feladat megoldásához módosítanom kellet a 2. feladatomban megírt függvényt továbbá a Viewban hozzá kellett adnom egy újabb TextFieldet és aktiváltam a Commit gombot. Az új Textfieldben a járat számát várja a program. A program annyiban változott, hogy a 2. feladatban megírt Update vagy Insert után egyből egy újabb tranzakció történik. Melyhez a fenti SQL kódot használom és ismét PreparedStatment-et használtam a feladatmegoldásához. A paraméterek az előző tranzakcióban létrehozott gép kódja és a hozzá rendelt járatnak a száma. Amennyiben a második tranzakció esetén hiba történik akkor egy rollback segítségével visszaállítjuk az első tranzakció lefutása előtti állapotba az adatbázist. Ebben a feladatban az automatikus commit-ot ki kell kapcsolni. Így csak a mi általunk megírt Commit gomb segítségével lehet commit-ot végezni. A hibákról ismét értesítjük a felhasználókat.

Mintaadatok:

KÓD	*TÍPUS*	*MUSZ_ELL*	*JAVITAS*	*LEGCSA-VAROS*	*ULES*	*HATOTAV*	*Megjegy-zés*	*JARAT-SZAM*
223	*B445B*	*2014-04-05*	*2013-03-05*	*1*	*420*	*4000*	*Test1*	*12*
11	*B445B*	*2014-04-08*	*2014-03-02*	*0*	*300*	*18000*	*Test2*	*400*

4.2 feladat

Magyarázat

Amikor az alkalmazást bezárom commit nélkül, akkor az előtte lévő tranzakció hatása nem fog érvényesülni. Ez azért van, mert amikor az autoCommit kivan kapcsolva, akkor egy connection.close() esetén nem érvényesülnek a nem commitolt tranzakciók, hanem egy rollback történik. Ezt úgy lehet megoldani, hogy visszakapcsoljuk az autocommitot, mert ekkor minden tranzakció után commitol egyet és ekkor ha elvesztjük a kapcsolatot vagy kilépünk akkor már a tranzakcióink érvényesülni fognak az adatbázison.

5 feladat

A megoldáshoz használt SQL utasítás

```
select count(*) from jarat a,  menetrend b
where a.szam=b.szam and a.honnan='PRG' and (exists (
select * from jarat, menetrend
where jarat.szam = menetrend.szam and hova='HEL'
and a.hova=jarat.honnan and
(bitand(b.nap,?)=bitand(menetrend.nap,?) or
a.hova = 'HEL')))
```

Magyarázat

A feladat megoldásához a model getStatistics(int bit) függvényét írtam meg. Ezt a függvényt meghívom minden alkalommal egy bitnek a helyiértékvel. Ez az int lesz a paramétere e fenti SQL utasításomnak, melyből megkapom az adott napra a jaratok számát PRG-ből HEL-be. Természetesen itt is PreparedStatement-et használtam. Az SQL lekérdezés eredményét egy ResultSetbe rakom bele, majd ezt visszaadom. Ezután a controllerben a ResultSetből getString() segítségével kiveszem az adott napra eső járatok számát és bele rakom egy tömbbe a nap nevével együtt, majd a stringet hozzáadom egy Listához. Miután megvizsgáltam egy for ciklus segítségével, hogy az egyes napokra hány járat jut átlagot számítok, és ezt hozzáadom a lista véghez. A viewban pedig a listának az elemeit kiírom a panelre.

6. Vélemény(ek) a mérésről

Vélemény, építő jellegű kritika.