Assignment #3

CS 348 - Spring 2015

Due on Saturday, July 25, 2015, 9 AM

For instructions on how to submit your assignment check the course website.

Question 1.

Total 40 p.

Consider the following Java class structure, where each comented line will be replaced by the contents of the files described below.

```
// 9_imports.txt
public class JDBCExample {
  public static void main(String[] args) throws Exception {
    // 1_driver.txt
    // 2_connect.txt
    // 3_autocommit.txt
    // 4_create.txt
    // 5_insert.txt
    // 6_procedure.txt
    // 7_select.txt
    // 8_close.txt
  }
}
```

Write the code described in each of the following files:

1_driver.txt – **1 p.** the file contains **a single line** with the Java code for loading the PostgreSQL driver.
Answer :

```
Class.forName("org.postgresql.Driver");
```

2_connect.txt – **1 p.** the file contains **a single line** with the Java code for connecting to the a database having:

- host : **localhost**
- port : **5432**
- database name : **postgres**
- username : **postgres**
- password : **postgres**

Answer :

```
Connection connection = DriverManager.getConnection
    ("jdbc:postgresql://127.0.0.1:5432/postgres", "postgres", "postgres");
```

3_autocommit.txt – **1 p.** the file contains **a single line** with the Java code for setting the auto commit to **false** for the established connection.
Answer :

```
connection.setAutoCommit(false);
```

4_create.txt – **3 p.** the file contains all the code necessary (including variable declarations) to perform **a single transaction** to define the tables *emp*, *dept* and *works*. The SQL statements for creating these tables are listed below and available on the course website. Your code **must use** a *Statement* object and make **3 calls** to the *executeUpdate* method.

```
CREATE TABLE emp
  (
     eid    NUMERIC(9, 0) PRIMARY KEY,
     ename  VARCHAR(30),
     age    NUMERIC(3, 0),
     salary NUMERIC(10, 2)
  );

CREATE TABLE dept
  (
     did       NUMERIC(2, 0) PRIMARY KEY,
     dname     VARCHAR(20),
     budget    NUMERIC(10, 2),
     managerid NUMERIC(9, 0) REFERENCES emp(eid)
  );

CREATE TABLE works
  (
     eid      NUMERIC(9, 0) REFERENCES emp,
     did      NUMERIC(2, 0) REFERENCES dept,
     pct_time NUMERIC(3, 0),
     PRIMARY KEY(eid, did)
  );
```

Answer :

```
Statement stmt = connection.createStatement();

String tabEmp = "CREATE TABLE emp ( eid NUMERIC(9, 0) PRIMARY KEY,"
    + " ename VARCHAR(30), age NUMERIC(3, 0),"
    + " salary NUMERIC(10, 2) );";
String tabDept = "CREATE TABLE dept ( did NUMERIC(2, 0) PRIMARY KEY,"
    + " dname VARCHAR(20), budget NUMERIC(10, 2),"
    + " managerid NUMERIC(9, 0) REFERENCES emp(eid) );";
String tabWorks = "CREATE TABLE works ( eid NUMERIC(9, 0) REFERENCES emp,"
    + " did NUMERIC(2, 0) REFERENCES dept, pct_time NUMERIC(3, 0),"
    + " PRIMARY KEY(eid, did) );";

stmt.executeUpdate(tabEmp);
stmt.executeUpdate(tabDept);
stmt.executeUpdate(tabWorks);

connection.commit();

stmt.close();
```

5_insert.txt – **5 p.** the file contains all the code necessary (including variable declarations) to perform **a single transaction** to populate the *emp* table with the tuples given in the file **emp.txt** available on the course website. The file will be given as a command-line argument to the class (*args[0]*). It

is the only command-line argument used. Each tuple in the file must be inserted as a separate insert statement (i.e. your SQL insert statement must use placeholders in order to perform batch insertions). Your code **must use** a *PreparedStatement* object.

Answer :

```
String insertEmp = "INSERT INTO emp ( eid, ename, age, salary) "
    + "VALUES (?, ?, ?, ?)";
PreparedStatement insertStmt = connection.prepareStatement(insertEmp);

File file = new File(args[0]);
BufferedReader br = new BufferedReader(new FileReader(file));
String line = null;

while ((line = br.readLine()) != null) {

    String[] lineSplit = line.split(",");

    insertStmt.setInt(1, Integer.parseInt(lineSplit[0]));
    insertStmt.setString(2, lineSplit[1]);
    insertStmt.setInt(3, Integer.parseInt(lineSplit[2]));
    insertStmt.setInt(4, Integer.parseInt(lineSplit[3]));

    insertStmt.executeUpdate();
}

br.close();
connection.commit();
insertStmt.close();
```

6_procedure.txt – **2 p**. the file contains all the code necessary (including variable declarations) to perform **a single transaction** to store the function *getnames(real)* in the database. The SQL statement for creating the function is listed below and available on the course website. Your code **must use** a *Statement* object.

```
CREATE OR REPLACE FUNCTION getnames(minsalary real)
RETURNS refcursor AS
$BODY$
DECLARE mycurs refcursor;
BEGIN
  OPEN mycurs FOR
  SELECT DISTINCT ename
  FROM            emp
  WHERE           salary >= minsalary
  ORDER BY        ename ASC;
  RETURN mycurs;
END
$BODY$
LANGUAGE plpgsql;
```

Answer :

```
Statement procStmt = connection.createStatement();
procStmt.execute("CREATE OR REPLACE FUNCTION getnames(minsalary real) "
    + "RETURNS refcursor AS "
    + "$BODY$ DECLARE mycurs refcursor; BEGIN OPEN mycurs FOR "
    + "SELECT DISTINCT ename FROM emp "
    + "WHERE salary >= minsalary ORDER BY ename ASC; "
    + "RETURN mycurs; END $BODY$ LANGUAGE plpgsql;");
connection.commit();
stmt.close();
```

7_select.txt – **5 p.** the file contains all the code necessary (including variable declarations) to perform **a single transaction** to call the function *getnames(real)* with the parameter 39000 and list the returned tuples on the standard out (*System.out*). To call the function your code **must use** a *CallableStatement* object.

Answer :

```
CallableStatement callStmt
    = connection.prepareCall("{ ? = call getnames(?) }");
callStmt.registerOutParameter(1, Types.OTHER);
callStmt.setInt(2, 39000);
callStmt.execute();

ResultSet results = (ResultSet) callStmt.getObject(1);
ResultSetMetaData rsmd = results.getMetaData();

int numberOfColumns = rsmd.getColumnCount();

while (results.next()) {
    for (int i = 1; i <= numberOfColumns; i++) {
       System.out.println(results.getString(i) + "\t");
    }
}

connection.commit();
results.close();
callStmt.close();
```

8_close.txt – **1 p.** the file contains **a single line** with the Java code for closing the database connection. Closing statements and result sets **should have already been done** in the file where they were declared and used.

Answer :

```
connection.close();
```

9_imports.txt – **1 p.** the file contains **all the imports necessary for the class to run**.

Answer :

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
```

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.sql.Types;
```

Some additional considerations:

- The class described above must be a runnable Java class once the content of each file will be copied in its designated place. The process of copying the files will be done automatically. It is your duty to ensure that the class can be executed. Therefore follow the specifications to the letter and test the class using the website resources before you submit it. The execution test will be worth half the marks for this exercise.
- Exception testing, while important, is not the focus of this exercise. The *throws Exception* is sufficent at this point. However you must make sure that with correct parameters (existing database, user, password, proper file path etc.) your class works.
- Do not submit any code in addition to what is required. You are, however, encouraged to take the application futher to learn more about accessing a database using JDBC.

Testing total 20 p.

```
cat 9_imports.txt class_top.txt 1_driver.txt
    2_connect.txt 3_autocommit.txt 4_create.txt
    5_insert.txt 6_procedure.txt 7_select.txt
    8_close.txt class_bottom.txt > JDBCExample.java

javac JDBCExample.java

java -cp 'postgresql-9.4-1201.jdbc4.jar:.' JDBCExample emp.txt
```

- **8 p.** class runs without exceptions
- **3 p.** the tables *emp*, *dept* and *works* appear in database after execution
- **3 p.** *count(*)* of *emp* is correct (i.e. 59 tuples)
- **3 p.** the procedure *getnames* appears in the database
- **3 p.** the program output displays the 28 names of *emp* with a salary above *39000*

0/8 p. for execution testing if the class does not run initially, but runs after correcting 2_connect.txt

0/20 p. if the class does not run even after correcting 2_connect.txt

Question 2.

Total 20 p.

Part I. We call a transaction that only reads database object a **read-only** transaction, otherwise the transaction is called a **read-write** transaction. Give brief answers to the following questions:

1. **2 p.** When does lock thrashing occur?
   Answer :

   Locking thrashing occurs when the database system reaches a point where adding another new active transaction actually reduces throughput due to competition for locking among all active transactions.

   Empirically, locking thrashing is seen to occur when 30% of active transactions are blocked.

2. **1 p.** If the database system *has not reached* the the thrashing point and the number of *read-write transactions* is increased, will the database throughput increase or decrease? (answer using one word)
   Answer :

   increase

3. **1 p.** If the database system *has reached* the the thrashing point and the number of *read-write transactions* is increased, will the database throughput increase or decrease? (answer using one word)
   Answer :

   decrease

4. **1 p.** If the number of *read-only transactions* is increased, will the database throughput increase or decrease? (answer using one word)
   Answer :

   increase

5. **3 p.** List three ways of tuning your system to increase transaction throughput.
   Answer :

   1. By locking the smallest sized objects possible.
   2. By reducing the time that transaction hold locks.
   3. By reducing hot spots, a database object that is frequently accessed and modified.

Part II. Consider the following schema:

Suppliers(*sid*: integer, *sname*: string, *address*: string)
Parts(*pid*: integer, *pname*: string, *color*: string)
Catalog(*sid*: integer, *pid*: integer, *cost*: real)

The *Catalog* relation lists the prices charged for parts by *Suppliers*.
For each of the following transactions, **state the SQL isolation level** that you would use and explain **in one sentence** why you chose it.

1. **3 p.** A transaction that adds a new part to a suppliers catalog.
   Answer :

   ```
   READ UNCOMMITTED
   We are inserting a new row in the table Catalog,
   we do not need any lock on the existing rows.
   ```

2. **3 p.** A transaction that increases the price that a supplier charges for a part.
   Answer :

   ```
   READ COMMITTED
   We are updating one existing row in the table Catalog and
   we need an exclusive lock on the row which we are updating.

   OR

   REPEATABLE READ
   If the update is done based on the current price,
   the price must not change during the transaction.
   ```

3. **3 p.** A transaction that determines the total number of items for a given supplier.
   Answer :

   ```
   SERIALIZABLE
   To prevent other transactions from inserting or updating the table Catalog
   while we are reading from it (known as the phantom problem).
   ```

4. **3 p.** A transaction that shows, for each part, the supplier that supplies the part at the lowest price.
   Answer :

   ```
   SERIALIZABLE
   To prevent other transactions from inserting or updating the table Catalog
   while we are reading from it (known as the phantom problem).
   ```

Question 3.

Total 40 p.

Part I. Consider a relation R with five attributes $ABCDE$.
You are given the following dependencies: $A \to B$, $BC \to E$, and $ED \to A$.

1. **6 p.** List all keys for R (as groups of attributes separated by comma, e.g. A, BC, DEF).
   Answer :

   ```
   Candidate keys: CDE, ACD, BCD
   Correct answer if superkeys are also listed.
   ```

2. **3 p.** Is R in 3NF? Explain your answer in one sentence.
   Answer :

   ```
   R is in 3NF because B, E and A are all parts of keys.
   ```

3. **3 p.** Is R in BCNF? Explain your answer in one sentence.
   Answer :

   ```
   R is not in BCNF because none of A, BC and ED contain a key.
   ```

Part II. Consider the attribute set $R = ABCDEGH$ and the FD set

$$F = AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G$$

1. For each of the listed attribute sets corresponding to your **section number**, do the following:

   (i) Compute the set of dependencies that hold over the set and write down a minimal cover.

   (ii) Name the strongest normal form that is not violated by the relation containing these attributes.

   (iii) Decompose it into a collection of BCNF relations if it is not in BCNF.

   Each set is worth **6 p.**

   $$Section\ 1 : (a)ABC, (c)ABCEG, (e)ACEH$$
   $$Section\ 2 : (a)ABC, (b)ABCD, (d)DCEGH$$

   Answer :

   ```
   (a) ABC

   (i) dependencies & minimal cover: AB->C, AC->B, BC->A

   (ii) normal form: BCNF
   # since AB, AC and BC are candidate keys for ABC

   (iii) decomposition: N/A
   ```

```
(b) ABCD

(i) dependencies & minimal cover: AB->C, AC->B, B->D, BC->A

(ii) normal form: 1NF
# the keys are: AB, AC, BC.

(iii) decomposition: ABC, BD

(c) ABCEG

(i) dependencies & minimal cover: AB->C, AC->B, BC->A, E->G

(ii) normal form: 1NF
# the keys are: ABE, ACE, BCE.

(iii)
decomposition: ABE, ABC, EG
# other decompositions are also possible

(d) DCEGH

(i) dependencies & minimal cover: E->G

(ii) normal form: 1NF
# the key is DCEH

(iii) decomposition: DCEH, EG

(e) ACEH

(i) dependencies & minimal cover: none

(ii) normal form: BCNF
# the key is ACEH itself

(iii) decomposition: N/A
```

2. Is the following decomposition of $R = ABCDEG$, with the same set of dependencies F, is
   (a) **5 p.** dependency-preserving?
   (b) **5 p.** lossless-join?

$$Section\ 1 : AB, BC, ABDE, EG$$

Answer :
(a) dependency-preserving

```
No.

Reason:
This decomposition does not preserve the FDs: AB -> C, AC -> B.
```

(b) lossless-join

No.

Reason:
The join between AB and BC is lossy. B is not a key for either one.

Section 2 : $ABC, ACDE, ADG$

Answer :
(a) dependency-preserving

No.

Reason:
ABC : AB -> C, AC -> B and BC -> A
ACDE : AD -> E
ADG : AD -> G (by transitivity)

The closure of this set of dependencies does not contain E -> G nor B -> D.
So this decomposition is not dependency preserving.

(b) lossless-join

Yes.

Reason:
The join of ABC and ACDE is lossless because their (attribute) intersection
is AC which is a key for ABCDE.
Joining this intermediate join with ADG is also lossless because the attribute
intersection is is AD and AD -> ADG.