# 11 Java JDBC

# Problema 1

Given:

```
Connection con = DriverManager.getConnection(dbURL);
con.setAutoCommit(false);
String updateString =
      "update SALES " +
      "set T_AMOUNT = 100 where T_NAME = 'BOB'";
Statement stmt = con.createStatement();
stmt.executeUpdate(updateString);
//INSERT CODE HERE
```

What statement should be added to the above code so that the update is committed to the database?

You had to select 1 option

- stmt.commit();
- con.commit();
    - Since auto-commit has been disabled in the given code (by calling con.setAutoCommit(false)), you have to call commit() on Connection object explicitly to commit the changes to the database.
- stmt.commit(true);
- con.commit(true);
- No code is necessary.

# Problema 2

You want to use a third party JDBC driver for a database. Which of the following actions must you take to retrieve a Connection using that driver in your JDBC program?

You had to select 2 options

- Put the driver jar in the class path.
- Load the driver class using Class.forName
- Create an instance of the Driver class using the new keyword.
- Retrieve a connection using DriverManager.getConnection.
- Load the driver class using Class.forName and then retrieve a connection using DriverManager.getConnection.

Explanation
Applications no longer need to explicitly load JDBC drivers using `Class.forName()`.
For example, to load the `my.sql.Driver` class, the `META-INF/services/java.sql.Driver` file would contain the entry:

```
my.sql.Driver
```

# Problema 3

Given:

```
Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample", "app",
"app");
Statement stmt = c.createStatement();
ResultSet rs = stmt.executeQuery("select * from STUDENT");
```

Which additional classes from java.sql package will you use if you want to find out how many columns are returned by the query?

You had to select 1 option

- MetaData
- ResultSetMetaData
- ColumnMetaData
- ResultMetaData

Explanation

`ResultSetMetaData` gives you the information about the result of executing a query. You can retrieve this object by calling getMetaData() on ResultSet.

`ResultSetMetaData` contains several methods that tell you about the ResultSet. Some important methods are:

getColumnCount(), getColumnName(int col), getColumnLabel(int col), and getColumnType(int col). Remember that the column index starts from 1.

# Problema 4

Identify the correct statement regarding a JDBC Connection:

You had to select 1 option

- When a JDBC Connection is created, it is in auto-commit mode.
- When a JDBC Connection is created, its commit mode depends on the parameters used while creating the connection.
- When a JDBC Connection is created, its auto-commit feature is disabled.
- When a JDBC Connection is created, it is in commit mode is undetermined.

Explanation:
When a connection is created, it is in auto-commit mode. i.e. auto-commit is enabled. This means that each individual SQL statement is treated as a transaction and is automatically committed right after it is completed. (A statement is completed when all of its result sets and update counts have been retrieved. In almost all cases, however, a statement is completed, and therefore committed, right after it is executed.)

The way to allow two or more statements to be grouped into a transaction is to disable the auto-commit mode. Since it is enabled by default, you have to explicitly disable it after creating a connection by calling `con.setAutoCommit(false);`

# Problema 5

You are using a RDBMS database from a company named Fandu Tech. It provides a JDBC 4.0 compliant driver implementation in class com.fandu.sql.Driver.

Which of the following lines of code is/are required to get the driver loaded?

You had to select 1 option

- Connection c = DriverManager.getConnection("jdbc:fandu://localhost:1234/myDB", "user", "pwd"); (Assume that the parameters are valid.)
    - This code is used to get a connection from the driver. It doesn't load the driver class itself.
- Class.forName("com.fandu.sql.Driver");
    - This is required for JDBC 1.3 and older versions. Not for JDBC 4.0.
- com.fandu.sql.Driver d = com.fandu.sql.Driver.class.newInstance()
- DriverManager.loadDriver("com.fandu.sql.Driver");
- None of the above.
    - In JDBC 4.0, the drivers are loaded automatically based on the information provided by the driver's META-INF/services/java.sql.Driver file. Therefore, no java code is necessary to load the driver classes.

# Problema 6

Identify the correct statements:

You had to select 2 options

- One advantage of CallableStatement is that it runs on the database.
    - All queries run on the database.
- One advantage of CallableStatement is that it allows IN/OUT parameters.
- One advantage of CallableStatement is that it uses Java instead of native SQL.
    - All queries executed on the database have to run in native SQL, whether fired through Statement, PreparedStatement, or from within a stored procedure. (Some database systems do allow Java code to run on the database as well, but that is beyond the scope of this exam.)
- A CallableStatement is the only way for a JDBC program to execute stored procedures in the database if the procedure has in and out parameters.

# Problema 7

Identify the correct statements:

You had to select 1 option

- A CallableStatement is used to call a stored procedure in the database while a PreparedStatement is used to call a stored function.
    - There is no difference in a stored procedure and function in terms of how it is called from JDBC code. A Callable is used for both. PreparedStatement is not used for either of these. It is used to call a regular SQL query repeatedly with different parameter values.

- A CallableStatement is easier to build and call from JDBC code than a PreparedStatement.
    - This is true because you don't have to write any SQL query in Java code. You just use the name of the stored procedure. The queries are already there inside the stored procedure, which exists in the Database and not in JDBC code.

- A CallableStatement is preferred over PreparedStatement because it supports long running transactions.
    - Even a PreparedStatement or a Statement can participate in a transaction (long or short). So this is not a reason to prefer Callable over PreparedStatement.

- A CallableStatement is preferred over Statement because it supports long running transactions.

# Problema 8

Which of the following illustrates a good coding practice given that the database connection is to be reused?

You had to select 1 option

```
try(Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery("select * from STUDENT"); )
{
   while(rs.next()){
     //do something with the row
   }
}
```

`Statement` and `ResultSet` objects are created in try-with-resources section and so there is no need to close them explicitly. Further, since connection is to be reused for other queries, it need not be closed either.

Remember that resources are closed at the end of the try block (and before any catch or finally) in the reverse of order of their creation. So here, `rs.close()` will be called first, followed by `stmt.close()`.

```
try(Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery("select * from STUDENT"); ) {
   while(rs.next()){
     //do something with the row
   }
}
finally{
  rs.close();
  stmt.close();
}
```

This code will not even compile because rs and stmt are not in scope for the finally block. They are in scope only within the try block.

```
try(Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery("select * from STUDENT"); ) {
   while(rs.next()){
     //do something with the row
   }
}
catch(SQLException e){
}
```

An empty catch block is not a good coding practice.

```
Statement stmt = null;
ResultSet rs = null;
try{
    stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery("select * from STUDENT");
   while(rs.next()){
     //do something with the row
   }
}
finally{
     rs.close(); stmt.close();
}
```

While not an entirely bad practice, it has a minor issue. If rs.close() throws an exception, stmt.close() will not be invoked. It is therefore always a good idea to use the auto close feature of the try-with-resources statement, which does not have this drawback.

# Problema 9

Consider the following code:

```
        Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample",
"app", "app");
        Statement stmt = c.createStatement();
        ResultSet rs = stmt.executeQuery("select * from CUSTOMER_ORDER"); //LINE 10
        stmt.close(); //LINE 11
        while(rs.next()){  //LINE 12
            System.out.println(rs.getString("AMOUNT"));  //LINE 13
        }
        c.close();
```

Assuming that CUSTOMER_ORDER table has multiple records and column AMOUNT is of type FLOAT, what will this code print?

(Assume that items not specified such as import statements  and try/catch block are all valid.)

You had to select 1 option

- It will throw an exception at //LINE 11

- It will throw an exception at //LINE 12
    - If the ResultSet object that you are trying to access is already closed, you will get an SQLException saying: Exception in thread "main" java.sql.SQLException: ResultSet not open.  In this case, although we are not closing the ResultSet directly, we are closing the Statement object from which the ResultSet was retrieved and when a Statement object is closed, its current ResultSet object, if one exists, is also closed.

- It will throw an exception at //LINE 13

- It will the print AMOUNT values
    - If the Statement object was not closed at //LINE 11, it would have printed the AMOUNT values.
    - Note that even though AMOUNT column is of type FLOAT, getString will convert the value to a String. An exception will not be thrown because of this.

# Problema 10

Consider the following code :

```
     Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery("select * from CUSTOMER where
EMAILID='bob@gmail.com'"); //LINE 10
        while(rs.next()){
            System.out.println(rs.getString("emailid")); //LINE 12
        }
        connection.close();
```

Assuming that the query returns exactly 1 row, what will be printed when this code is run?
(Assume that items not specified such as import statements, DB url, and try/catch block are all valid.)

You had to select 1 option

- It will throw an exception at //LINE 12.
- Compilation will fail due to //Line 12
- It will print `bob@gmail.com`
- It will print bob@gmail.com and then throw an exception.

Explanation
1. "select *" implies you are selecting all the columns.
2. You can retrieve the values from a ResultSet using either the columns indices, which starts with 1 or using the actual column names of the database tables.
3. Column names used as input to getter methods are case insensitive. When a getter method is called with a column name and several columns have the same name, the value of the first matching column will be returned.

Now, it is given that the query returns 1 row, therefore, it can be assumed that EMAILID is a valid column name in this table. Therefore, there is no problem with the code and it should print the value 'bob@gmail.com'.

# Problema 11

Assuming that the table student2 does not have any row, how many rows will be present in this table after the following code is executed?

```
        try (Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample",
"app", "app");) {
            c.setAutoCommit(false);
            Statement stmt = c.createStatement();
            int a1 = stmt.executeUpdate("insert into STUDENT2 values (1, 'aaa', 1.1)");
            Savepoint sp1 = c.setSavepoint();
            int a2 = stmt.executeUpdate("insert into STUDENT2 values (2, 'bbb', 2.1)");
            c.rollback();
            int a3 = stmt.executeUpdate("insert into STUDENT2 values (3, 'ccc', 3.1)");
            c.setAutoCommit(true);
            int a4 = stmt.executeUpdate("insert into STUDENT2 values (4, 'ddd', 4.1)");
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

You had to select 1 option

- 1
- 2
- 3
- 4
- 0
- An exception stack trace will be printed.

Explanation

First, the connection's auto commit mode is set to false and then two rows are inserted. However, since autocommit is `false`, they are not committed yet.
Now, `c.rollback()` is called. This will cause the whole transaction to rollback. Although we created a `Savepoint`, we did not pass it to rollback and so the save point will have not be respected and both the rows will be lost. This transaction ends here.

Next, row 3 is inserted in a new transaction and since autocommit is still false at this point, it is not commited yet. The transaction is still running.
Now, when `c.setAutoCommit(true)` is called, the auto-commit status changes from false to true and this causes the existing transaction to commit thereby committing row 3.

Finally, row 4 is inserted and committed immediately because autocommit is true.

Therefore, the table will end up with 2 rows.

# Problema 12

Given the following RDBMS table information :

```
  STUDENT Table
      SID INT Primary Key
      NAME VARCHAR(50)
      GPA INT

and the following code:


        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery("select SID, NAME,  GPA from STUDENT");
        while(rs.next()){
            System.out.println(  INSERT CODE HERE );
        }
        connection.close();
```

What can be inserted in the above code so that it will print the GPA value for each student?

(Assume that items not specified such as import statements and try/catch block are all valid.)

You had to select 2 options

- rs.getString(2)
  - The numbering of columns in a ResultSet starts with 1. Therefore, it should be rs.getString(3).

- rs.getString(3)
  - Although the value of the GPA field is int, it can still be retrieved using getString(). Note that if a field is of type VARCHAR and if you try to retrieve the value using say getInt(), it may throw an exception at runtime if the value cannot be parsed into an Integer.

- rs.getInt(2)
  - The numbering of columns in a ResultSet starts with 1. Therefore, it should be rs.getInt(3).

- rs.getInteger(2)

- rs.getInt("GPA")

# Problema 13

What will the following code fragment print when compiled and run?

```
Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample", "app",
"app");
try(Statement stmt = c.createStatement();)
{

    ResultSet rs = stmt.executeQuery("select * from STUDENT");
    while(rs.next()){
        System.out.println(rs.getString(1));
    }

}
catch(SQLException e){
    System.out.println("Exception ");
}
```

(Assume that items not specified such as import statements and try/catch block are all valid.)

You had to select 1 option

- It will throw an exception if the first column of the result is not a String.
    - Since Object class has a toString() method, any type can be converted to a String. So this cannot be a cause of the exception. If you use getInt() or getDouble and if the column value cannot be converted to an int or double, then that will cause an Exception.

- It will throw an exception every time it is run irrespective of what the query returns.

- It will print the value for the first column of the result and if there is no row in STUDENT table, it will not print anything.
    - rs.next() returns true only if there is a row left to be processed in the ResultSet. It moves the cursur in front of the next row and returns true. If there is no row left, it returns false.
    - Thus, when you use rs.next() in a while loop, you are basically iterating through the rows returned by the query one by one. rs.getString(1) returns the value of the first column of the row.

- It will not compile.

# Problema 14

Given the following RDBMS table information :

```
  STUDENT Table
      SID INT Primary Key
      NAME VARCHAR(50)
      GPA INT

and the following code:


        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery("select SID, NAME,  GPA from STUDENT where
SID=10");
        while(rs.next()){
            System.out.println( rs.getString(1) );
        }
        connection.close();
```

What will it print when run?

(Assume that items not specified such as import statements and try/catch block are all valid.)

You had to select 1 option

- It will print 10 if the row with SID 10 exists.

- It will print the NAME of the student whose SID is 10.
    - Since the numbering of columns in a ResultSet starts with 1, getString(1) will return SID.
    - Although the value of the SID field is INT, it can still be retrieved using getString().
    - Note that if a field is of type VARCHAR and if you try to retrieve the value using say getInt(), it may throw an exception at runtime if the value cannot be parsed into an Integer.

- It will print SID of all the Students after 10.
    - Since SID is the Primary key of the table, there can be at most one row with SID 10 in the table.

- It will print NAME of all the Students having SID equal to 10.

# Problema 15

Given:

```
Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample", "app",
"app");
try(Statement stmt = c.createStatement();)
{

    ResultSet rs = stmt.executeQuery("select * from STUDENT");
    ResultSetMetaData rsmd = rs.getMetaData();
    int cc = rsmd.getColumnCount();
    while(rs.next()){
        for(int i = 0; i<cc; i++){
            System.out.print(rsmd.getColumnName(i)+" = "+rs.getObject(i+1)+", ");
        }
        System.out.println();
    }

}
catch(SQLException e){
    e.printStackTrace();
}
```

What will the above code print?
(Assume that there is data in Student table.)

You had to select 1 option

- It will print an exception stack trace at run time.
    - Since column indexing starts with 1, rsmd.getColumnName(0) will cause an exception. Other than this, there is no issue with the code.
- It will print all the rows and columns returned by the query.
- It will print all the columns for the first row and then print an exception stack trace.
- It will print all but the last row and then print an exception stack trace.
- It will not compile.

# Problema 16

The following is a partial code listing from an application:

```
rs.moveToInsertRow();
rs.updateString(1, "111");
rs.updateString(2, "Java in 24 hours");

//INSERT CODE HERE
```

Assuming that rs is a valid reference to an updatable ResultSet and that the result set contains only two String columns, what can be inserted in the code so that a new row is inserted in the database?

You had to select 1 option

- rs.updateRow();
  - updateRow is used when you are trying to update an existing row. If you call this method while the cursor is on the insert row, an SQLException will be thrown.
- rs.insertRow();
- rs.commit();
  - ResultSet does not have commit method.
- rs.insert();
  - ResultSet does not have insert() method.

Explanation

JDBC 2.0 allows you to use ResultSet object to update an existing row and even insert new row in the database. For both the cases, the ResultSet must be updatable, which can be achieved by passing `ResultSet.CONCUR_UPDATABLE` while creating a `Statement` object:

```
stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

or `stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);`

The general usage pattern for this functionality is as follows -
To update an existing row:

1. First, go to the row you want to update. You can either iterate through a `ResultSet` to reach a particular row or just call `rs.absolute(int rowNumber).`
2. Now update the columns of the `ResultSet` with required values using `rs.updateXXX(columnNumber, value)` or `rs.updateXXX(columnName, value)` methods.
3. Call `rs.updateRow();` If you call `rs.refreshRow()` without calling `updateRow()`, the updates will be lost.

To insert a new Row:

1. Call `rs.moveToInsertRow();` first. You can't insert a row without calling this method first.
2. Use `rs.updateXXX` methods to update all column values. You must set values for all the columns.
3. Call `rs.insertRow();`
4. Call `rs.moveToCurrentRow();` to go back to the row where you were before calling moveToInsertRow.

# Problema 17

A java.sql.Statement object …

You had to select 1 option

- implements a session with the database.
    - A java.sql.Connection is meant for that.

- provides a cursor to fetch data from the database.
    - A java.sql.ResultSet object does that.

- provides a means to execute queries on the database.
    - Statement provides various execute methods to execute queries on the database.

- provides methods to commit and rollback transactions.
    - A java.sql.Connection object does that.

# Problema 18

Objects of which of the following classes/interfaces are independent of the implementation of a JDBC driver?

You had to select 4 options

- java.sql.Connection
- java.sql.Statement
- java.sql.ResultSet
- javax.sql.RowSet
- java.sql.Date
- java.sql.Time
- java.sql.SQLException

Explanation
The actual classes for Connection, Statement, and ResultSet interfaces are provided by the JDBC driver and are therefore driver dependent.

The RowSet interface is unique in that it is intended to be implemented using the rest of the JDBC API. In other words, a RowSet implementation is a layer of software that executes "on top" of a JDBC driver. Implementations of the RowSet interface can be provided by anyone, including JDBC driver vendors who want to provide a RowSet implementation as part of their JDBC products. Oracle provides sample implementations of various RowSet interfaces with the JDK as well. So it is not clear whether this option is a correct option or not but we have included it because a few candidates have reported getting such a question.

# Problema 19

Consider the following code written to generate a report containing customer data where each line is a pipe separated list of values.

```
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery("select EMAILID, NAME,  PHONE from CUSTOMER
order by EMAILID"); //LINE 10
        while(rs.next()){
            System.out.println(rs.getString(0)+"|"+rs.getString(1)+"|"+rs.getString(3));
        }
        connection.close();
```

Identify the correct statement about this code.
(Assume that items not specified such as import statements and try/catch block are all valid.)

You had to select 1 option

- It will throw an exception when run if there is at least 1 record in the table.
    - The numbering of columns in a ResultSet starts with 1. Therefore, rs.getString(0) will cause an SQLException.
- It will throw an exception when run if there are no records in the table.
- It will print the data as expected.
- It will not compile.

# Problema 20

What will the following code fragment print when compiled and run, given that the table t_account has 10 rows?

```
Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample", "app",
"app");
try(Statement stmt = c.createStatement();)
{
    stmt.setMaxRows(2);
    ResultSet rs = stmt.executeQuery("select id, accnum, name from t_account");
    while(rs.next()){
        System.out.println(rs.getString(1));
    }

}
catch(SQLException e){
    System.out.println("Exception ");
}
```

(Assume that items not specified such as import statements and try/catch block are all valid.)

You had to select 1 option

- It will print two ids from the table.
- It will print two accnums from the table.
- It will print all ten ids from the table.
- It will print all ten accnums from the table.
- It will not compile.
- It will throw an exception at run time.

Explanation
1. You should remember that column index starts from 1. Therefore, getString(1) will return the value of the first column i.e. the id column.

2. getString retrieves the value of the designated column in the current row of this ResultSet object as a String. If the value is SQL NULL, the value returned is null.

3. Statment has a setMaxRows method that limits the total number of rows returned by the ResultSet. If the query returns more rows than the limit, the extra rows are silently (i.e. without any exception) are ignored.

# Problema 21

Which of the following method would you use to get a JDBC connection while using JDBC 4.0 but not while using a previous version of JDBC?
You had to select 2 options

- ```
  public Connection getConnection1(String url, String userid, String pwd) throws Exception{
      Properties p = new Properties();
      p.setProperty("user", userid);
      p.setProperty("password", pwd);
      return DriverManager.getConnection(url, p);
  }
  ```
  - This is a valid way in JDBC 4.0 to get a connection.

- ```
  public Connection getConnection2(String url, String userid, String pwd) throws Exception{
      Properties p = new Properties();
      p.setProperty("user", userid);
      p.setProperty("password", pwd);
      DriverManager.registerDriver("com.xyz.Driver");
      return DriverManager.getConnection(url, p);
  }
  ```
  - There are 3 problems with it:
  - 1. DriverManager.registerDriver is a valid method but it takes java.sql.Driver instance and not a String.
  - 2. This method is used by the Driver class to register itself with the DriverManager. It need not be called by the application programmer.
  - 3.In JDBC 4.0, if you have the jar file that implements the Driver in the classpath, the Driver is automatically registered.

- ```
  public Connection getConnection3(String url, String userid, String pwd) throws Exception{
      Properties p = new Properties();
      p.setProperty("jdbc.driver", "com.xyz.Driver");
      p.setProperty("jdbc.user", userid);
      p.setProperty("jdbc.user.password", pwd);
      return DriverManager.getConnection(url, p);
  }
  ```
  - 1. The property names jdbc.user and jdbc.user.password are invalid. They should be user and password.
  - 2. jdbc.driver is also an invalid property name but it is not an error to use it. It will not be used.

- ```
  public Connection getConnection4(String url, String userid, String pwd) throws Exception{
      Class.forName("com.xyz.Driver");
      return DriverManager.getConnection(url, userid, pwd);
  }
  ```

- Loading a Driver class explicitly is required only in lower versions of JDBC and not for JDBC 4.0.

```
public Connection getConnection5(String url, String userid, String pwd) throws SQLException{
  return DriverManager.getConnection(url, userid, pwd);
}
```

# Problema 22

Identify the correct statements regarding JDBC.

You had to select 1 option

- The JDBC Driver must be loaded explicitly in the code before attempting to create a connection to the database.

- Starting JDBC 4.0, the JDBC Driver class is not required any more.

- Starting JDBC 4.0, the JDBC Driver class is not required to be loaded explicitly in the code any more.
  - To load a JDBC driver in JDBC 1.3, the application code would have to load the Driver class explicitly using Class.forName method, for example - Class.forName("com.xyz.jdbc.Driver"). However, with JDBC 4.0, applications no longer need to do this.

- JDBC 4.0 allows loading multiple JDBC Drivers while previous versions did not.
  - There has never been any restriction on how many JDBC Drivers are allowed to be loaded by an application. If your application connects to say 3 different databases, you can load three different Drivers. Of course, starting JDBC 4.0, you don't need to load the Driver classes explicitly in the code, while earlier, you had to explicitly load them using Class.forName.

# Problema 23

Consider the following code:

```
    Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample",
"app", "app");
        Statement stmt = c.createStatement();
        ResultSet rs = stmt.executeQuery("select * from STUDENT");
        stmt.close();
        while(rs.next()){
            System.out.println(rs.getInt(1));
        }
        c.close();
```

Assuming that STUDENT table has 2 records contains values 1 and 2 for the first column, what will this code print?
(Assume that items not specified such as import statements  and try/catch block are all valid.)

You had to select 1 option

- 1
  2

- 2

- Exception at run time
  - If the ResultSet object that you are iterating is closed, you will get an exception saying: Exception in thread "main" java.sql.SQLException: ResultSet not open.
  - In this case, although we are not closing the ResultSet directly, we are closing the Statement object from which the ResultSet was retrieved and when a Statement object is closed, its current ResultSet object, if one exists, is also closed.

- Compilation failure

# Problema 24

You have a long running process that does computations in multiple steps and saves the values computed at each to the database. In some situations, the process may encounter a special condition and may be required to recompute the values starting from a certain step depending on the condition.

What JDBC method call will you use to efficiently manage this requirement?
(Assume that the variable names used in the statement below refer to valid objects of the type implied by their names.)

You had to select 1 option

- connection.commit(false);
- connection.rollback();
- connection.rollback(true);
- connection.rollback(savePoint);

Explanation

The java.sql.Savepoint interface of JDBC is tailor made for this purpose. At the end of each step, the process can create a save point. Later on, when the process encounters the special condition, it can roll back the transaction up to any of the previous save point and continue from there, instead of rolling back the whole transaction and losing all the values. For example,

```
connection.setAutoCommit(false);
stmt.executeUpdate("update student set status=1"); //1
Savepoint savePoint1 = connection.setSavepoint("step1done");
stmt.executeUpdate("update student set gpa=4.0"); //2
if(special condition){
  connection.rollback(savePoint1);
}
connection.commit(); //query 1 will be committed but query 2 will not be
committed.
```