Chapter 12

JDBC
(Java DataBase Connectivity)

Business Training

1

1Z0-809 Chapter 12

Content

- Introduction to JDBC
- Connecting to a Database
- Submitting Queries and Reading Results from the Database
- Updating the Database

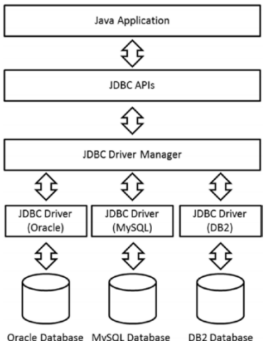
2

Introduction to JDBC

3

What is JDBC ?

- **JDBC (Java Database Connectivity) is an important Java API that defines how a client accesses a database. As such, it is critical in building large-scale enterprise Java solutions.**
  - At a high level, interacting with a database involves the following steps:
    1. Establish a connection to a database.
    2. Execute SQL queries to retrieve, create, or modify tables in the database.
    3. Close the connection to the database
- **The JDBC classes and interfaces are part of the packages java.sql.* and javax.sql.***
- **A simplified architecture of JDBC is represented in this Figure**
  - JDBC drivers and the driver manager play a key role in realizing the objective of JDBC.
  - JDBC drivers are specifically designed to interact with their respective DBMSs.



4

Setting up a Database

- **Before you begin exploring JDBC APIs and their usage, you must set up a database with which to work.**
  1. Download the latest MySQL installer from the MySQL download page (www.mysql.com/downloads/mysql).
  2. Invoke the MySQL installer, and follow all the steps shown by the installation wizard. Keep the default values, and complete the installation. The installer asks you to provide a root/admin password; remember it, because it's used in the examples.
  3. Invoke the MySQL command-line client (in our case, it is MySQL 8.0 Command Line Client, shown on the Start menu). You see a MySQL prompt once you provide the root/admin password.
     - Optionnaly install a GUI Client for MySQL like MySQL Workbench from this link https://dev.mysql.com/downloads/workbench/

5

Connecting to a Database

6

The Connection Interface

- **The Connection interface of the java.sql package represents a connection from application to the database. It is a channel through which your application and the database communicate.**

Method	Description
`Statement createStatement()`	Creates a `Statement` object that can be used to send SQL statements to the database.
`PreparedStatement prepareStatement(String sql)`	Creates a `PreparedStatement` object that can contain SQL statements. The SQL statement can have IN parameters; they may contain ? symbol(s), which are used as placeholders for passing actual values later.
`CallableStatement prepareCall(String sql)`	Creates a `CallableStatement` object for calling stored procedures in the database. The SQL statement can have `IN` or `OUT` parameters; they may contain ? symbol(s), which are used as placeholders for passing actual values later.
`DatabaseMetaData getMetaData()`	Gets the `DataBaseMetaData` object. This metadata contains database schema information, table information, and so on, which is especially useful when you don't know the underlying database.
`Clob createClob()`	Returns a `Clob` object (`Clob` is the name of the interface). Character Large Object (CLOB) is a built-in type in SQL; it can be used to store a column value in a row of a database table.
`Blob createBlob()`	Returns a `Blob` object (`Blob` is the name of the interface). Binary Large Object (BLOB) is a built-in type in SQL; it can be used to store a column value in a row of a database table.
`void setSchema(String schema)`	When passed the schema name, sets this `Connection` object to the database schema to access.
`String getSchema()`	Returns the schema name of the database associated with this `Connection` object; returns `null` if no schema is associated with it.

Connecting to the Database Using DriverManager

- **The first step in communicate with your database is to set up a connection between your application and the database server. Establishing a connection requires understanding the database URL, so let's discuss it now.**
  - Here is the general format of the JDBC URL:**jdbc:<subprotocol>:<subname>**
    - An example of a URL string is **jdbc:mysql://localhost:3306/MyDB**
    - The following program acuires a connection

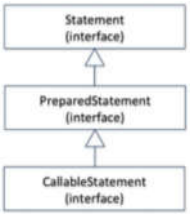
```
class DbConnect {
    public static void main(String[] args) {
         // URL points to JDBC protocol: mysql subprotocol;
         // localhost is the address of the server where we installed our
         // DBMS (i.e. on local machine) and 3306 is the port on which
         // we need to contact our DBMS
          String url = "jdbc:mysql://localhost:3306/";

        // we are connecting to the addressBook database we created earlier
        String database = "addressBook";
        // we login as "root" user with password "mysql123"
        String userName = "root";
        String password = "mysql123";

        try (Connection connection = DriverManager.getConnection
                (url + database, userName, password)){
                 System.out.println("Database connection: Successful");
        } catch (Exception e) {
               System.out.println("Database connection: Failed");
               e.printStackTrace();
        }
    }
}
```

Submitting Queries and Reading Results from the Database

9

Statement Interface

- **As the name suggests, Statement is a SQL statement that can be used to communicate a SQL statement to the connected database and receive results from the database.**
  - You can form SQL queries using Statement and execute it using APIs provided in the Statement interface (or one of its derived interfaces)



- **JDBC Statements**
  - **Statement**: Sends a SQL statement to the database without any parameters. For typical uses, you need to use this interface. You can create an instance of Statement using the createStatement() method in the Connection interface.
  - **PreparedStatement**: Represents a precompiled SQL statement that can be customized using IN parameters. Usually, it is more efficient than a Statement object; hence, it is used to improve performance, especially if a SQL statement is executed multiple times. You can get an instance of PreparedStatement by calling the preparedStatement() method in the Connection interface.
  - **CallableStatement**: Executes stored procedures. CallableStatement instances can handle IN as well as OUT and INOUT parameters. You need to call the prepareCall() method in the Connection interface to get an instance of this class

10

Statement Interface

- **Important Methods of the Statement Interface**

Method	Description
`boolean execute(String sql)`	Executes the given SQL query. This method returns true if the query resulted in a `ResultSet`. You can retrieve the `ResultSet` object by calling the `getResultSet()` method. This method returns false if the SQL query has no results or if there is an update count. You can use the `getUpdateCount()` method to get the update count. In rare situations, this method may return multiple `ResultSets`; in that case, you can call the `getMoreResults()` method.
`ResultSet executeQuery(String sql)`	Executes the query and returns the `ResultSet` object as the result. If there are no results, the method does not return `null`; rather, the returned `ResultSet` object will return false when the `next()` method is called.
`int executeUpdate(String sql)`	Executes `CREATE`, `INSERT`, `UPDATE`, or `DELETE` SQL queries. It returns the number of rows updated (or zero if there is no result, such as with the `CREATE` statement).
`Connection getConnection()`	Returns the `Connection` object with which the `Statement` object was created.
`void close()`	Closes the database and other JDBC resources associated with this `Statement` object. Calling `close()` on an already-closed `Statement` object has no effect.

11

ResultSet Interface

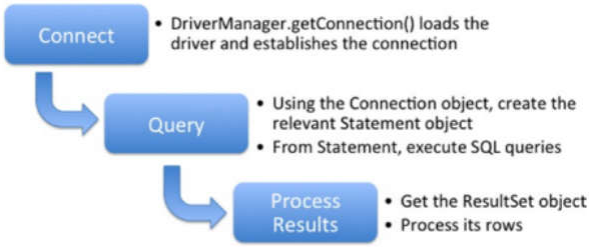
- **A resultset maintains a cursor pointing to the current row.**
  - You can read only one row at a time, so you must change the position of the cursor to read/navigate through the entire resultset. Initially, the cursor is set to just before the first row.
    - You need to call the next() method on the resultset to advance the cursor position by one row. This method returns a Boolean value; hence you can use it in a while loop to iterate over the entire resultset

Method	Description
`void beforeFirst()`	Sets the cursor just before the first row in the resultset.
`void afterLast()`	Sets the cursor just after the last row of the resultset.
`boolean absolute(int rowNumber)`	Sets the cursor to the requested row number (absolute position in the table—not relative to the current position).
`boolean relative(int rowNumber)`	Sets the cursor to the requested row number relative to the current position. `rowNumber` can be a positive or negative value: a positive value moves forward, and a negative value moves backward relative to the current position.
`boolean next()`	Sets the cursor to the next row of the resultset.
`boolean previous()`	Sets the cursor to the previous row of the resultset.

12

Querying the Database

- **Now you know all the necessary interfaces that are used to execute a simple SQL query on a database: Connection, Statement, and ResultSet. Figure below shows the high-level steps for establishing connection to the database, executing SQL queries, and processing the results**



Querying the database

- The following code snippet queries all records from the *Users* table and print out details for each record:

```
String sql = "SELECT * FROM Users";

Statement statement = conn.createStatement();
ResultSet result = statement.executeQuery(sql);

int count = 0;

while (result.next()){
    String name = result.getString(2);
    String pass = result.getString(3);
    String fullname = result.getString("fullname");
    String email = result.getString("email");

    String output = "User #%d: %s - %s - %s - %s";
    System.out.println(String.format(output, ++count, name, pass, fullname, email));
}
```

Updating the Database

15

Inserting new rows

- **We create a parameterized SQL INSERT statement and create a PreparedStatement from the Connection object.**
  - To set values for the parameters in the INSERT statement, we use the **PreparedStatement's** setString()methods because all these columns in the table Users are of type VARCHAR which is translated to String type in Java.
    - Note that the parameter index is 1-based (unlike 0-based index in Java array).

```
String sql = "INSERT INTO Users (username, password, fullname, email) VALUES (?, ?, ?, ?)";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "bill");
statement.setString(2, "secretpass");
statement.setString(3, "Bill Gates");
statement.setString(4, "bill.gates@microsoft.com");

int rowsInserted = statement.executeUpdate();
if (rowsInserted > 0) {
    System.out.println("A new user was inserted successfully!");
}
```


16

Updating the database

- **The following code snippet will update the record of "Bill Gates" as we inserted previously.**

```
String sql = "UPDATE Users SET password=?, fullname=?, email=? WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "123456789");
statement.setString(2, "William Henry Bill Gates");
statement.setString(3, "bill.gates@microsoft.com");
statement.setString(4, "bill");

int rowsUpdated = statement.executeUpdate();
if (rowsUpdated > 0) {
    System.out.println("An existing user was updated successfully!");
}
```


Deleting rows

- **The following code snippet will delete a record whose *username* field contains "bill":**

```
String sql = "DELETE FROM Users WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "bill");

int rowsDeleted = statement.executeUpdate();
if (rowsDeleted > 0) {
    System.out.println("A user was deleted successfully!");
}
```

JDBC Hints

- **The actual classes for Connection, Statement, and ResultSet interfaces are provided by the JDBC driver and are therefore driver dependent.**
  - The RowSet interface extends the standard java.sql.ResultSet interface and is implemented as part of JDBC driver.
- **A DB Connection is always in auto-commit mode when it is created.**
- **You can pass Properties object o jdbc connection:**

```
Properties p = new Properties();
p.setProperty("user", userid);
p.setProperty("password", pwd);
Connection c =  DriverManager.getConnection(dburl, p);
```

- **updateRow is used when you are trying to update an existing row.**

```
given:  stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
rs.moveToInsertRow()
rs.updateString(1, "111");
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. Otherwise, use: rs.insertRow();

JDBC Hints

- **createStatement(ResultSet TYPE, ResultSet CONCURRENT mode)**

  The ResultSet type parameter is passed to createStatement() before the concurrency mode.
  - There are three "ResultSet type" options: **TYPE_FORWARD_ONLY**, **TYPE_SCROLL_INSENSITIVE**, and **TYPE_SCROLL_SENSITIVE**.
  - There are two ResultSet concurrency modes: **CONCUR_READ_ONLY** and **CONCUR_UPDATABLE**. All database drivers support read-only result sets, but not all support updatable ones.
    If you request options that the database driver does not support, it downgrades to an option it does support rather than throwing an exception.
- **SavePoint feature of JDBC.**
  - 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.
```

JDBC Hints

- **The execute() method is allowed to run any type of SQL statements. The executeUpdate() method is allowed to run any type of the SQL statement that returns a row count rather than a ResultSet**
- **The requirement to include a java.sql.Driver file in the META-INF directory was introduced in JDBC 4.0. Older drivers are not required to provide it. All drivers are required to implement the Connection interface.**
- **You can call stmt over and over, its ok to re-use the rs instance like below...**

```
rs = stmt.executeQuery
rs = stmt.executeQuery
etc.
```

  - When running a query on a Statement, Java closes any already open ResultSet objects!!
- **ResultSet get functions are 1-based!**
  - rs.relative(-5) requires the ResultSet be scrollable
  - rs.relative(-x) cant move farther back than 0
- **Scrollable result set does not loop around.**
  - BUT: rs.absolute(-1) always puts cursor at last record in result set
  - ResultSet options need to be supplied when creating the Statement object rather than when executing the query.

JDBC Hints

- **When creating the Statement, the code doesn't specify a result set type. This means it defaults to TYPE_FORWARD_ONLY.**
  - The absolute() method can only be called on scrollable result sets. The code throws a SQLException
  - ResultSet options need to be supplied when creating the Statement object rather than when executing the query.