DATABASE PROGRAMMING WITH JDBC

Instructor: DieuNT1

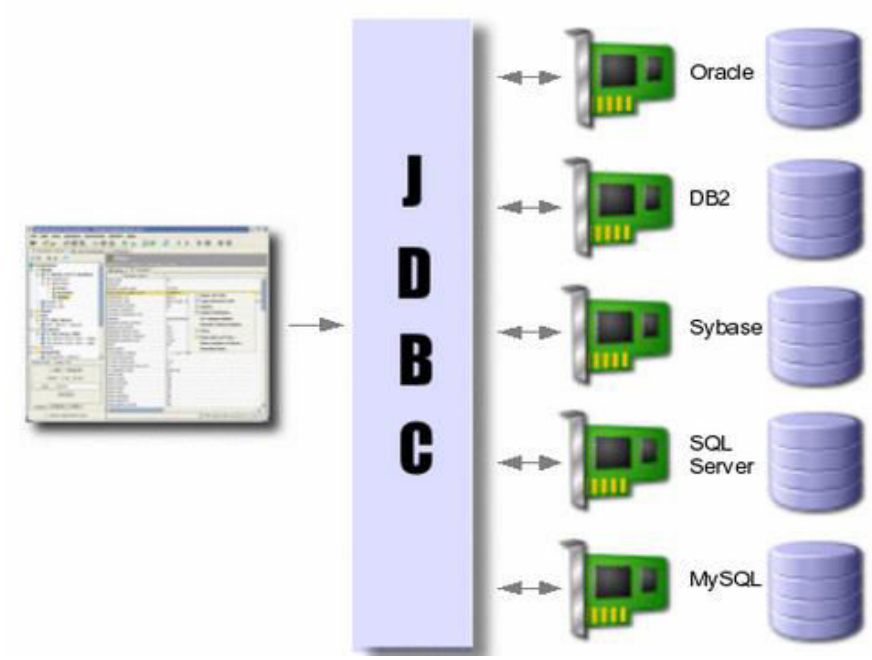
Table of contents

Section 1

JAVA JDBC TUTORIAL

Overview

- JDBC (Java Database Connectivity) API allows Java programs to connect to databases
- Database access is the same for all database vendors
- The JVM uses a JDBC driver to translate generalized JDBC calls into vendor specific database calls.

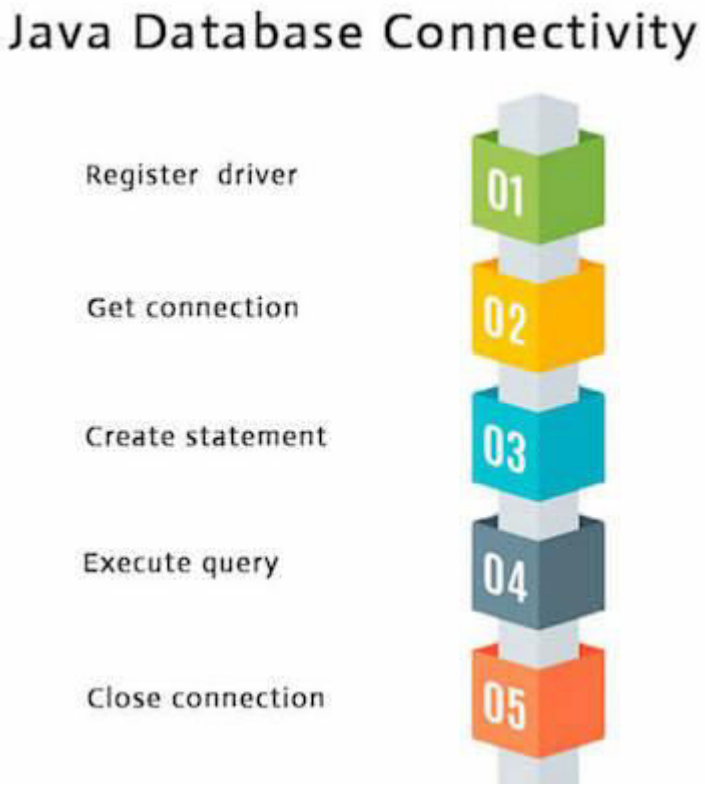


Section 2

WORKING STEPS

Working steps

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection



Register the driver class

- The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.
- Example to register the **OracleDriver** class:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

- Example to register the **SQLServerDriver** class:

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

- Example to register the **MySqlServerDriver** class:

```
Class.forName("com.mysql.jdbc.Driver");
```

- ***Note**: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.*

Create Connection

- The **getConnection()** method of DriverManager class is used to establish connection with the database.
- Syntax of getConnection() method:
  - ✓ `public static Connection getConnection(String url)throws SQLException`
  - ✓ `public static Connection getConnection(String url,String name,String password) throws SQLException`

- Example to establish connection with the **Oracle** database

```
Connection con=DriverManager.getConnection(
                    "jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

- Example to establish connection with the **My SQL Server** database

```
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/ebookshop?
          allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC",
          "myuser", "xxxx");
```

- Example to establish connection with the **MS SQL Server** database

```
String connectionUrl = "jdbc:sqlserver://localhost:1433;databaseName=Fsoft_Training";
Connection conn = DriverManager.getConnection(connectionUrl, "system","password");
```

Create Access Statement

- The **createStatement**() method of **Connection** interface is used to create statement. The object of statement is responsible to execute queries with the database.
  - ✓ Use for general-purpose access to your database.
  - ✓ Useful when you are using static SQL statements at runtime.
  - ✓ The **Statement** interface cannot accept parameters.
- **Syntax:**

```
Statement stmt = null;
try {
  stmt = conn.createStatement(); // or
  stmt = con.createStatement(ResultSetType,
                             ConcurencyType);
} catch (SQLException e) {
}
finally { stmt.close(); }
```

Execute the query

- The **executeQuery**() method of **Statement** interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.
- Syntax of **executeQuery**() method:
  - ✓ `public ResultSet executeQuery(String sql)throws SQLException`
- **Example**:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM EMP");

while(rs.next()){
    System.out.println(rs.getInt(1) + " " + rs.getString(2));
}
```

Close the connection object

- By closing connection object statement and **ResultSet** will be closed automatically.
- The close() method of **Connection** interface is used to close the connection.
- Syntax of **close**() method:

```
public void close() throws SQLException
```

- **Example**:

```
con.close();
```

- ***Note***: *Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.*

Section 3

DRIVERMANAGER CLASS

DriverManager class

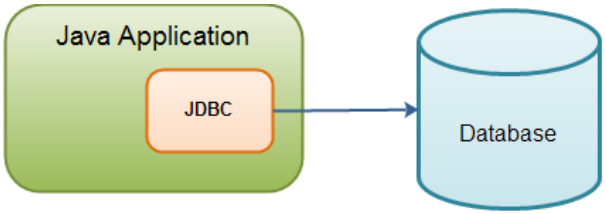
- The **DriverManager** class acts as an interface between **user** and **drivers**. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- The **DriverManager** class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().
- **Methods**:

Method	Description
1) public static void **registerDriver**(Driver driver)	is used to register the given driver with DriverManager.
2) public static void **deregisterDriver**(Driver driver)	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection **getConnection**(String url)	is used to establish the connection with the specified url.
4) public static Connection **getConnection**(String url, String userName, String password)	is used to establish the connection with the specified url, username and password.

Connection interface

- A **Connection** is the session between **Java application** and **database**.
- The Connection interface is a factory of **Statement**, **PreparedStatement** and **DatabaseMetaData**.
- *By default, connection commits the changes after executing queries.*
- **Methods**:
  - ✓ public Statement **createStatement()**: creates a statement object that can be used to execute SQL queries.
  - ✓ public Statement **createStatement**(int resultSetType, int resultSetConcurrency):

    creates a Statement object that will generate ResultSet objects with the given type and concurrency.
  - ✓ public void **setAutoCommit**(boolean status): is used to set the commit status. By default it is **true**.



Section 4

JDBC STATEMENT

Statement interface

- The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.
- **Create Statement:**

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

Statement statement = connection.createStatement(
        int resultSetType, int resultSetConcurrency);

Statement statement = connection.createStatement(
        int resultSetType, int resultSetConcurrency,
        int resultSetHoldability)
```

Statement interface

- Statement's **methods**:
  - ✓ **boolean execute(String SQL)** : may be any kind of SQL statement. Returns a boolean value of true if a ResultSet object can be retrieved; false if the first result is an update count or there is no result.
  - ✓ **int executeUpdate(String SQL)** : Returns the numbers of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
  - ✓ **ResultSet executeQuery(String SQL)** : Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.
  - ✓ **public int[] executeBatch():** is used to execute batch of commands.

Examples

- **Example 1: Execute a SELECT query via a Statement**

```
// Create and execute an SQL statement that returns some data.
String SQL1 = "SELECT TOP 10 * FROM Person";
Statement stmt=conn.createStatement();
//ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
ResultSet rs = stmt.executeQuery(SQL);
```

- **Example 2: Execute an INSERT via a Statement**

```
// Create and execute an SQL statement that returns some data.
String SQL2 = "INSERT INTO STOCK(STOCK_CODE, STOCK_NAME)
          VALUES('11', 'STOCK1')";
Statement stmt = conn.createStatement();
int no_of_row = stmt.executeUpdate(SQL);
```

Retrieve Data & Close Connection

- **Retrieve data**

```
// Iterate through the data in the result set and display it.
while (rs.next()) {
  System.out.println(rs.getInt(1) + "\t" +
                     rs.getString(2)+"\t"+rs.getInt(3));
}
```

- **Close connection**

```
statement.close();
conn.close();
```

Statement Using Java Try With Resources

- In order to close a Statement correctly after use, you can open it inside a Java **Try With Resources** block.
- Here is an example of closing a Java JDBC Statement instance using the **try-with-resources** construct:

```
try (Statement statement = connection.createStatement()) {
      // use the statement in here.
} catch (SQLException e) {
      // TODO: handle exception
}
```

- *Once the try block exits, the **Statement** will be closed automatically.*

Section 5

JDBC RESULTSET

Overview

- The Java JDBC **java.sql.ResultSet** interface represents the result of a database query.
- This ResultSet is then iterated to inspect the result.
- **A ResultSet Contains Records:**
  - ✓ A JDBC ResultSet contains records. Each records contains a set of columns. Each record contains the same amount of columns, although not all columns may have a value. A column can have a null value.
  - ✓ The following ResultSet has 3 different columns (Name, Age, Gender), and 3 records with different values for each column

Name	Age	Gender
John	27	Male
Jane	21	Female
Jeanie	31	Female

ResultSet example - records with columns

Creating a ResultSet

- You create a **ResultSet** by executing a **Statement** or **PreparedStatement**, like this:

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

ResultSet result = statement.executeQuery("SELECT * FROM dbo.Course");
```

- Or like this:

```
String selectQuery = "SELECT * FROM dbo.Course";

PreparedStatement statement = connection.prepareStatement(selectQuery);

ResultSet result = statement.executeQuery();
```

- **ResultSet** data:

next()
next()
next()

	1	2	3	4	5
	course_id	subject_id	course_code	title	number_of_credits
1	11111	CSCI	1301	Introduction to Java I	4
2	11112	CSCI	1302	Introduction to Java II	3
3	11113	CSCI	3720	Database Systems	3
4	11114	CSCI	4750	Rapid Java Application	3
5	11115	MATH	2750	Calculus I	5
6	11116	MATH	3750	Calculus II	5
7	11117	EDUC	1111	Reading	3
8	11118	ITEC	1344	Database Administration	3

```
while (result.next()) {
        System.out.println(result.getString(1)
                + "\t" + result.getString(2)
                + "\t" + result.getString(3)
                + "\t" + result.getString(4)
                + "\t" + result.getInt(5));
        }
```

ResultSet Example

```
public List<Course> findCourseByName(String name) throws SQLException {

    Connection connection = null;
    Statement statement = null;
    ResultSet result = null;

    List<Course> courses = new ArrayList<Course>();

    try {

      connection = DBUtils.getConnection();

      statement = connection.createStatement(ResultSet.TYPE_FORWARD_ONLY,
          ResultSet.CONCUR_READ_ONLY);

      result = statement.executeQuery(
          "SELECT * FROM dbo.Course WHERE title LIKE '%" + name + "%'");

      Course course;
```

ResultSet Example

```
      while (result.next()) {
        course = new Course(result.getString(1), result.getString(2),
            result.getString(3), result.getString(4), result.getInt(5));

        courses.add(course);
      }
    } finally {
      if (statement != null) {
        statement.close();
      }

      if (result != null) {
        result.close();
      }
    }

    return courses;
  }
```

ResultSet Example

```
public class CourseTest {

  public static void main(String[] args) {
    CourseDao courseDao = new CourseDaoImpl();
    String name = "Java";

    try {
      List<Course> courses = courseDao.findCourseByName(name);
      courses.forEach(c -> System.out.println(c));

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

  }

}
```

JDBC Update using ResultSet

```
ResultSet rs = statement.executeQuery(query);
…
// for update
rs.updateBoolean(1, false); // change the first column
rs.updateInt("Age", 25); // change the column named "Age"
rs.updateRow();

// to delete
rs.deleteRow();
```

Section 6

JDBC PREPAREDSTATEMENT

(with Parameter)

PreparedStatement Interface

- The PreparedStatement interface extends the Statement interface which gives you added functionality with a couple of advantages over a generic Statement object.

```
public interface PreparedStatement extends Statement {

}
```

- It is used to execute parameterized query.
- **Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

PreparedStatement Interface

- The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.
- **Syntax**:

```
public PreparedStatement prepareStatement(String query)
                                        throws SQLException{}
```

- This statement gives you the flexibility of supplying **arguments dynamically**.

```
PreparedStatement pstmt = null;
try {
    String SQL = "Update Employees SET age = ? WHERE id = ?";
    pstmt = conn.prepareStatement(SQL);
} catch (SQLException e) {
    //TODO
} finally {
    //TODO
}
```

Methods of PreparedStatement interface

Method	Description
public void setInt(int paramIndex, int value)	Sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	Sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	Sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	Sets the double value to the given parameter index.
public int **executeUpdate**()	Executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	Executes the select query. It returns an instance of ResultSet.

JDBC With Parameter

- The **setXXX()** methods bind values to the parameters.

- **Examples**:

```
pstmt.setInt(1,23);
pstmt.setString(2,"Roshan");
pstmt.setString(3,"CEO");
pstmt.executeUpdate();
```

PreparedStatement Example

```java
public boolean save(Course course) throws SQLException {

    PreparedStatement preparedStatement = null;

    Connection connection = null;

    int result;
    try {
      connection = DBUtils.getConnection();

      String query = "INSERT INTO dbo.Course VALUES (?,?,?,?,?)";
      preparedStatement = connection.prepareStatement(query);

      preparedStatement.setString(1, course.getCourseId());
      preparedStatement.setString(2, course.getSubjectId());
      preparedStatement.setString(3, course.getCourseCode());
      preparedStatement.setString(4, course.getCourseTitle());
      preparedStatement.setInt(5, course.getNumOfCredits());

      result = preparedStatement.executeUpdate();

    } finally {
      if (preparedStatement != null) {
        preparedStatement.close();
      }
      if (connection != null) {
        connection.close();
      }
    }

    return (result > 0);
  }
```

PreparedStatement Example

```java
public static void main(String[] args) {
    CourseDao courseDao = new CourseDaoImpl();

    Course course = new Course("11119", "ITC", "1205",
        "Java SE Programming Language", 5);

    try {
      boolean resultSave = courseDao.save(course);

      System.out.println(resultSave);

    } catch (SQLException e1) {
      e1.printStackTrace();
    }

  }
```

Results:

```
true
```

Thank you