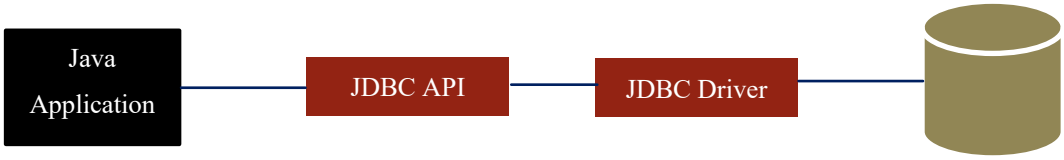
Contents

5.1 JDBC API (Java Database Connectivity)

- It allows Java programs to access relational databases.
- It provides various classes, methods, and interfaces that are helpful in easy communication with the database.



5.2 Common JDBC Components

- **DriverManager** – It loads database-specific drivers in an application and establishes a connection with a database.
- **JDBC Driver** – It help us to communicate with a data source through JDBC.
- **Connection** – This interface with all methods for contacting a database. All communication with database is through connection object only.
- **Statement** – It is an interface that represents a SQL statement.
- **SQLException** – This class handles any errors that occur in a database application.
- **ResultSet** – It is an interface that contains table data returned by a SELECT query.

5.3 Processing SQL Statements with JDBC

- In general, the following steps are needed –
    1. Establishing a connection
    2. Create Statement Object
    3. Write SQL query and execute the query
    4. Close the connection.

Establishing Connection

- Connection object can be established from getConnection () method of DriverManager class.
    1. getConnection(URL,username,password)
    2. getConnection(URL)

Some Connection String

Database	Connection URL
MySQL	jdbc:mariadb://HOST_NAME:PORT/DATABASE_NAME
PostgreSQL	jdbc:postgresql://HOST_NAME:PORT/DATABASE_NAME
Oracle	jdbc:oracle:thin:@HOST_NAME:PORT:SERVICE_NAME
Microsoft SQL Server	jdbc:sqlserver://HOST_NAME:PORT;DatabaseName=< DATABASE_NAME>

Example

```
//using method 1
try(Connection con = DriverManager
                .getConnection ("jdbc:mariadb://localhost:3306/test", "root","");){
        System.out.println("Create Connection Object");
}

//using method 2
try(Connection con = DriverManager
                .getConnection("jdbc:mariadb://localhost:3306/test?user=root")){
        System.out.println("Create Connection Object");
}
```

Creating Statement Object

- Three different kinds of statements:
    1. **Statement**
    2. **PreparedStatement**
    3. **CallableStatement**

Statement

- Used to implement simple SQL statements with no parameters.

Example:

```
Statement stm = con.createStatement();
```

PreparedStatement

- Used for parameterized and precompiling SQL statements.
- To add parameters to the PreparedStatement, we can use simple setters – setX() – where X is the type of the parameter.
- It also helps to prevent SQL Injection.

Example

```
String query = “UPDATE employee SET city =? WHERE id =?”;
PreparedStatement pst = con.prepareStatement(query);
pst.setString(1, "Mandalay");
pst.setInt(2, 1001);
```

CallableStatement

- Used to execute stored procedures.
- Setting input parameter values for the stored procedure is done like in the PreparedStatement.

Example – CallableStatement

```
// stored procedure
DELIMITER //
CREATE PROCEDURE saveEmployee(_id INT, _name VARCHAR(50),_salary DECIMAL)
BEGIN
    INSERT INTO employee(empNo,name,salary)VALUES(_id,_name,_salary);
END//

// using callableStatement
String query = "{call saveEmployee(?,?,?)}";
CallableStatement cstm = con.prepareCall(query);
cstm.setInt(1, 1003);
cstm.setDouble(3, 2000000);
cstm.setString(2, "aung aung");
```

Executing SQL Query

Four important methods to execute the query –

executeQuery()	For SELECT query Return one ResultSet object.
executeUpdate()	For updating the data (e.g. INSERT, DELETE, UPDATE) or DDL statements. Return the number of affected rows.
execute()	Can be used for both cases above when the result is unknown. Return true if it executes the SELECT query. Return false if it executes INSERT or UPDATE query. Use if the query could return one or more ResultSet objects.
executeBatch()	Used to execute the batch of SQL queries. If all the queries are successful then, it returns the array of update counts. It is mostly used to insert or update the records in bulk.

Example

```
//Employee Table
CREATE TABLE employee
(
empNo INT  PRIMARY KEY,
name VARCHAR(30) NOT NULL,
salary double,
city VARCHAR(30),
birthday Date
);
```

Example – executeUpdate (Inserting new data)

```
String url = "jdbc:mariadb://localhost:3306/test";
String username = "root";
String password = "";

try(Connection con = DriverManager.getConnection(url, username, password)){

        String insert = """
                        INSERT INTO employee(empNo,name,salary,birthday,city)
                        VALUES(?,?,?,?,?)
                """;
        // create statement
        PreparedStatement pst = con.prepareStatement(insert);
        // binding parameters
        pst.setInt(1, 1001);
        pst.setString(2, "Htet Htet");
        pst.setDouble(3, 1000000);
```

```
        LocalDate dob =LocalDate.of(1997, 9, 1);
        pst.setDate(4, Date.valueOf(dob));
        pst.setString(5, "Yangon");
        //execute INSERT query
        int result = pst.executeUpdate();
        System.out.println((result > 0) ? "Insert Success" : "Insert Fail");
}
```

Example – executeBatch

```
try(Connection con = DriverManager.getConnection(url, username, password);
        Scanner sc = new Scanner(System.in)){

        String insert = """
                        INSERT INTO employee(empNo,name,salary,birthday,city)
                        VALUES(?,?,?,?,?)
                """;
        PreparedStatement pst = con.prepareStatement(insert);
        for(var i = 1;i <= 3;i++) {
                System.out.print("Enter employee no: ");
                pst.setInt(1, sc.nextInt());
                sc.nextLine();
                System.out.print("Enter name: ");
                pst.setString(2, sc.nextLine());
                System.out.print("Enter salary: ");
                pst.setDouble(3, sc.nextDouble());
                sc.nextLine();
                System.out.print("Enter birthday(yyyy-MM-dd): ");
                pst.setDate(4, Date.valueOf(sc.nextLine()));
                System.out.print("Enter city: ");
                pst.setString(5, sc.nextLine());

                pst.addBatch();
        }
        pst.executeBatch();
        System.out.println("Insert Success....");
}
```

Employee Table

empNo	name	salary	city	birthday
1001	Htet Htet	1000000	Yangon	1997-09-01
1002	Kyaw Kyaw	5000000	Yangon	1995-01-01
1003	Aung Aung	500000	Mandalay	1992-10-16
1004	Kaung Kaung	600000	Monywa	1995-08-25
1005	Nay Nay	2000000	Yangon	2008-08-10

Example – executeQuery

```
String query1 = "SELECT * FROM employee";
PreparedStatement pst = con.prepareStatement(query1);
ResultSet rs = pst.executeQuery();
```

Example – executeUpdate(Updating & Deleting Data)

```
String update = """
                UPDATE employee SET salary = salary + (salary * 0.2)
                WHERE city = ?
        """;
PreparedStatement pst = con.prepareStatement(update);
pst.setString(1, "Yangon");
int result = pst.executeUpdate();
System.out.println("No of row updated: " + result);
```

```
String delete = "DELETE FROM employee WHERE name LIKE ?";
PreparedStatement pst = con.prepareStatement(delete);
pst.setString(1, "%aung%");
int result = pst.executeUpdate();
System.out.println("No of row deleted: " + result);
```

Closing Connection

- After performing various operations and manipulations on the database, its close method is needed to call in order to release the database resources.

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

- Alternatively, use a try-with-resources statement to automatically close.

5.4 How to retrieve data from ResultSet

- ResultSet object is used to access the data retrieved from the Database.
- It uses the next () method to move to the next row.
- Retrieving the value for each table cell can be done using methods of type getXX().

Example – Retrieve all rows

```
String query1 = "SELECT * FROM employee";
PreparedStatement pst = con.prepareStatement(query1);
ResultSet rs = pst.executeQuery();
while(rs.next()) {
        System.out.print(rs.getInt(1) + "\t");//using index
        System.out.print(rs.getString(2) + "\t");
        System.out.print(rs.getString("city") + "\t"); //using col name
        System.out.print(rs.getDouble("salary") + "\t");
        System.out.print(rs.getDate("birthday") + "\n");
}
```

Example –Retrieve specific rows

```
String query1 = "SELECT * FROM employee WHERE salary BETWEEN ? AND ?";
PreparedStatement pst = con.prepareStatement(query1);
pst.setDouble(1, 500000);
pst.setDouble(2, 1000000);
ResultSet rs = pst.executeQuery();
while(rs.next()) {
    // retrieve data
}
```

```
String query1 = "SELECT empNo,name,city FROM employee WHERE empNo IN (?,?,?)";
PreparedStatement pst = con.prepareStatement(query1);
pst.setInt(1, 1001);
pst.setInt(2, 1003);
pst.setInt(3, 1005);
ResultSet rs = pst.executeQuery();
while(rs.next()) {
   //retrieve data
}
```

```
String query1 = "SELECT * FROM users WHERE email = ? AND password = ?";
PreparedStatement pst = con.prepareStatement(query1);
pst.setString(1, "jk@gmail.com");
pst.setString(2, "jeon");
ResultSet rs = pst.executeQuery();
```

```
if(rs.next())
        System.out.println("Login success");
else
        System.err.println("Login fail");
```

5.5 Handling Transactions

- A transaction is a set of one or more statements that are executed as a unit.
- A database must satisfy the ACID properties to guarantee the success of a database transaction.
    - **Atomicity**: Each transaction should be carried out in its entirety; if one part of the transaction fails, then the whole transaction fails
    - **Consistency**: The database should be in a valid state before and after the performed transaction.
    - **Isolation**: Each transaction should execute in complete isolation without knowing the existence of other transactions.
    - **Durability**: Once the transaction is complete, the changes made by the transaction are permanent (even in the occurrence of unusual events such as power loss).

Transaction Management in JDBC

- The following steps are required for transaction management.
    1. Disabling Auto-Commit Mode
    2. Committing Transactions.
    3. Roll backing Transaction if any one of statements fails.

Typical Transaction Handling Workflow

```
try {

   // begin the transaction:
   connection.setAutoCommit(false);

   // execute statement
     ----------
     ----------
   // commit the transaction
   connection.commit();

} catch (SQLException ex) {
   // abort the transaction
   connection.rollback();
} finally {
   // close connection
 }
```

Example

Student Table

```
CREATE TABLE Students
(
 Id BIGINT PRIMARY KEY AUTO_INCREMENT,
 Name VARCHAR(100),
 Email VARCHAR(30) UNIQUE
);
```

```
Connection con = null;
try{
        con = DriverManager.getConnection(""
        + "jdbc:mariadb://localhost:3306/test","root","");

        String insert1 = "INSERT INTO students(Name,Email)VALUES(?,?)";
        String insert2 = "INSERT INTO students(Name,Email)VALUES(?,?)";
        String update = "UPDATE students SET Email = ? WHERE Id = ?";

        //step - 1
        con.setAutoCommit(false);

        //insert statement
        PreparedStatement insertPstm = con.prepareStatement(insert1);
        insertPstm.setString(1, "Jeon");
        insertPstm.setString(2, "jeon@gmail.com");
        insertPstm.executeUpdate();
```

```
            insertPstm = con.prepareStatement(insert2);
            insertPstm.setString(1, "Htet Htet");
            insertPstm.setString(2, "htet@gmail.com");
            insertPstm.executeUpdate();

            //update statement
            PreparedStatement updatePstm = con.prepareStatement(update);
            updatePstm.setString(1, "jeon@gmail.com");
            updatePstm.setInt(2, 2);
            updatePstm.executeUpdate();

            //step - 3
            con.commit();
}catch (Exception e) {
            if(con != null) {
                        //step - 2
                        System.err.println("Transaction has been rolled back");
                        con.rollback();
            }
}
                        finally {
                                    con.close();
                        }
```

Further Reading

1. https://www.codejava.net/java-se/jdbc/java-jdbc-transactions-tutorial
2. https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html
3. https://www.onlinetutorialspoint.com/jdbc/transaction-management-jdbc-example.html

Assignment 1

s

THANK YOU