SmartCliff
CAREER MOBILITY SOLUTIONS

We are on a mission to address the digital skills gap for 10 Million+ young professionals, train and empower them to forge a career path into future tech

Java Database Connectivity

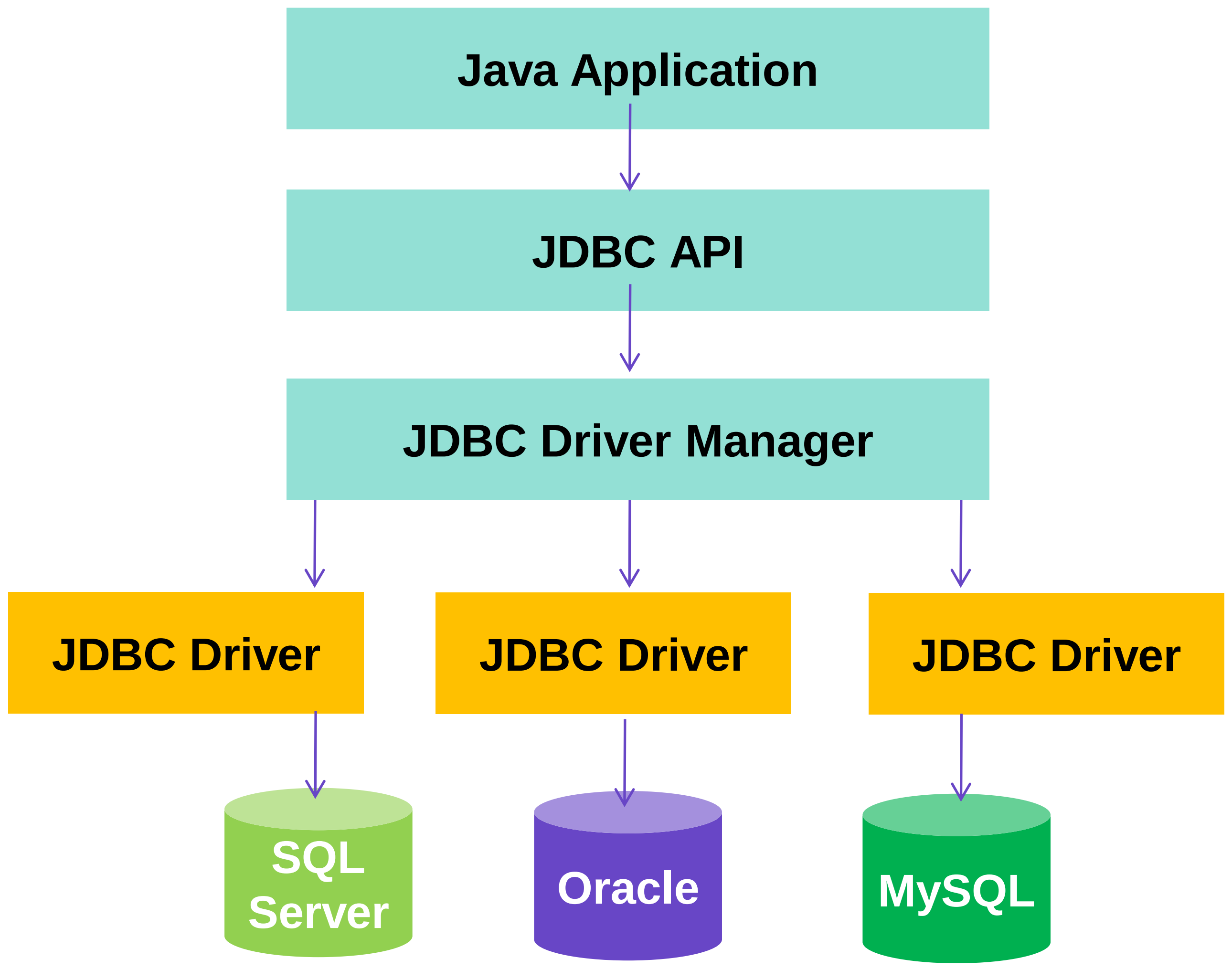
Introduction

- **JDBC** (Java Database Connectivity) is a Java-based data access technology from Oracle Corporation.
- JDBC is a **Java API** (Application Programming Interface). It provides methods for querying and updating data in a database.
- It is a part of **JavaSE** (Java Standard Edition).
- JDBC was initially conceived as a **client-side API**, enabling a Java client to interact with a data source
- JDBC release then has featured updates to both the **client-side package** (**java.sql**) and the **server-side package (javax.sql)**

Why JDBC ?

- Before JDBC, **ODBC API** was the **database API** to connect and execute the query with the database.
- But, ODBC API uses **ODBC driver** which is written in **C language** (i.e. platform dependent and unsecured).
- That is why Java has defined its **own API** (JDBC API) that uses JDBC drivers which is written in Java language.
- We can use JDBC API to handle database using **Java program**.

JDBC Architecture



JDBC Architecture

- JDBC offers a **programming-level interface** that handles the mechanics of Java applications communicating with a **database or RDBMS**.
- JDBC interface consists of **two layers:**
- **JDBC API** supports communication between the **Java application** and the **JDBC driver manager.**
- JDBC driver supports communication between the **JDBC driver manager** and the **database driver.**
- **JDBC Driver Manager**: The basic service for managing a set of JDBC drivers.

JDBC Drivers

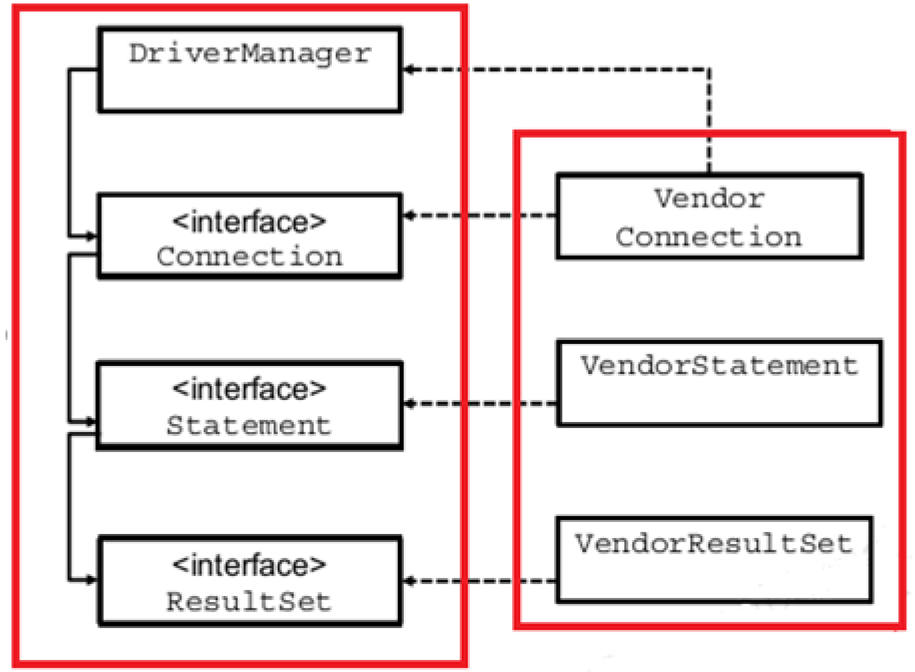
- There are **four types of JDBC drivers**

  1. JDBC-ODBC Bridge Driver **(Deprecated in Java 8 and later)**
  2. Native Driver
  3. Network Protocol Driver
  4. Thin Driver **(Most Commonly Used)**

Steps to Connect to the Database

There are **5 steps** to connect any java application with the database using JDBC

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection



Steps to Connect to the Database

- The JDBC **DriverManager class** defines objects which can **connect Java applications to a JDBC driver**.
- DriverManager is considered the **backbone** of JDBC architecture.
- DriverManager class **manages the JDBC drivers** that are installed on the system.
- Its **getConnection() method** is used to establish a connection to a database.

Steps to Connect to the Database

Steps 1) Register the driver class

- To load the driver or register it before using it in the program. There should be registration once in program. We can register a driver in any of the **two ways**:
  1. Using **forName()** method
  2. Using **registerDriver()** method

1. The **forName()** method of Class is used to **register the driver class.** This method is used to **dynamically load** the driver class. Once loaded, the Driver class creates an instance of itself.

Steps to Connect to the Database

Syntax:

```
public static void forName(String className)throws ClassNotFoundException
```

Example: Java program to **load oracle driver** to establish database connection

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

Steps to Connect to the Database

2. Using **registerDriver()** method

- DriverManager is an inbuilt class of Java that comes with a static member register.
- Need to call the drivers class' constructor at compile-time .

Syntax:

```
public static void registerDriver(Driver driver) throws SQLException
```

Example: Java program to **load oracle driver** to establish database connection

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

Steps to Connect to the Database

Steps 2) Create the connection object

- The **getConnection()** method of **DriverManager** class is used to **establish connection** with the database
- Uses a **username, password, and a jdbc url** to establish a connection to the database
- It returns a **connection object**.
- A jdbc Connection represents a **session/connection** with a specific database.
- Within the context of a Connection, the statements are **executed and results** are returned.
- An application can have **one or more** connections with a single database, or it can have **many connections** with different databases.

Steps to Connect to the Database

Syntax:

public static Connection getConnection(String url) throws SQLException

public static Connection getConnection(String url,String name,String password)

throws SQLException

Example:

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

Where **jdbc** is the API,

oracle is the database,

thin is the driver,

localhost is the server name on which oracle is running, we may also use IP address,

1521 is the port number and XE is the Oracle service name

Steps to Connect to the Database

Steps 3) Create the statement object

- The createStatement() method of **Connection interface** is used to create statement.
- The object of statement is responsible to execute queries with the database.
- Can execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set.
- Need a Connection object to create a Statement object

Syntax:

public Statement createStatement() throws SQLException

Steps to Connect to the Database

Example:

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

JDBC represents **statements** using one of the following classes:

Statement:

- Used to implement simple SQL statements with no parameters.

PreparedStatement: (Extends Statement.)

- The statement is cached and then the execution path is pre-determined on the database server allowing it to be executed **multiple times** in an efficient manner.

Steps to Connect to the Database

CallableStatement: (Extends PreparedStatement.)

- Used to execute **stored procedures** that may contain both input and output parameters

Steps 4) 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:

```
public ResultSet executeQuery(String sql)throws SQLException
```

Steps to Connect to the Database

Example:

```
        ResultSet rs=stmt.executeQuery("select * from employee");

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

Steps to Connect to the Database

Steps 4) Execute the Query

To execute a query, call an **execute method from Statement** such as the following:

execute:

- Returns **true** if the first object that the query returns is a ResultSet object.
- Use this method if the query could return one or moreResultSet objects.
- Retrieve the ResultSet objects returned from the query by repeatedly calling Statement.getResutSet.

executeQuery:

- Returns one ResultSet object.

executeUpdate:

- Returns an integer representing the **number of rows** affected by the SQL statement. Use this method if you are using **INSERT,DELETE, or UPDATE SQL** statements

Steps to Connect to the Database

Steps 5) Close the connection object

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

Syntax:

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

Example:

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

JDBC with Oracle

Information required about Oracle database as follows:

Driver class:

- The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.

Connection URL:

- The connection URL for the oracle11G database is **jdbc:oracle:thin:@localhost:1521:xe**

− Where jdbc is the API,

− oracle is the database,

− thin is the driver,

− localhost is the server name on which oracle is running, we may also use IP address,

− 1521 is the port number and XE is the Oracle service name

Username:

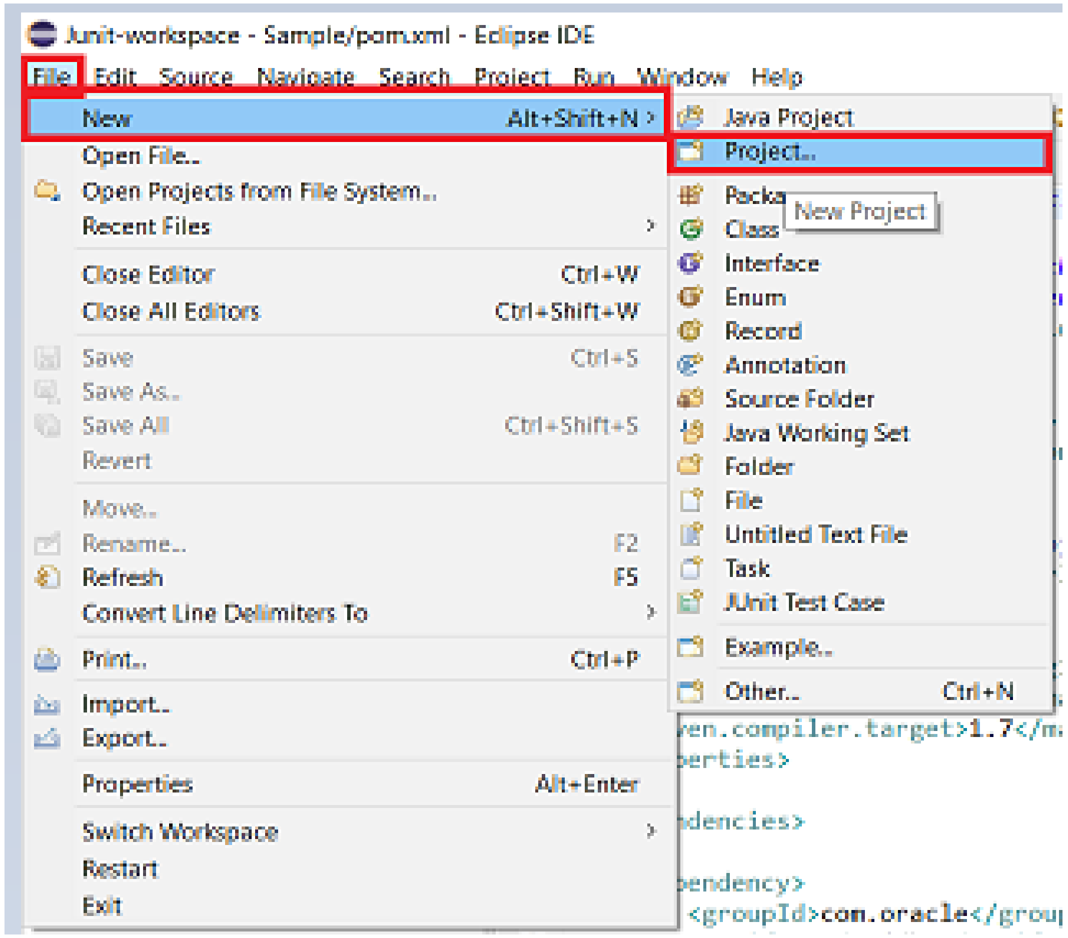
- The default username for the oracle database is system.

Password:

- Password given by the user at the time of installing the oracle database.

Configuring a JDBC development environment (Using Maven)

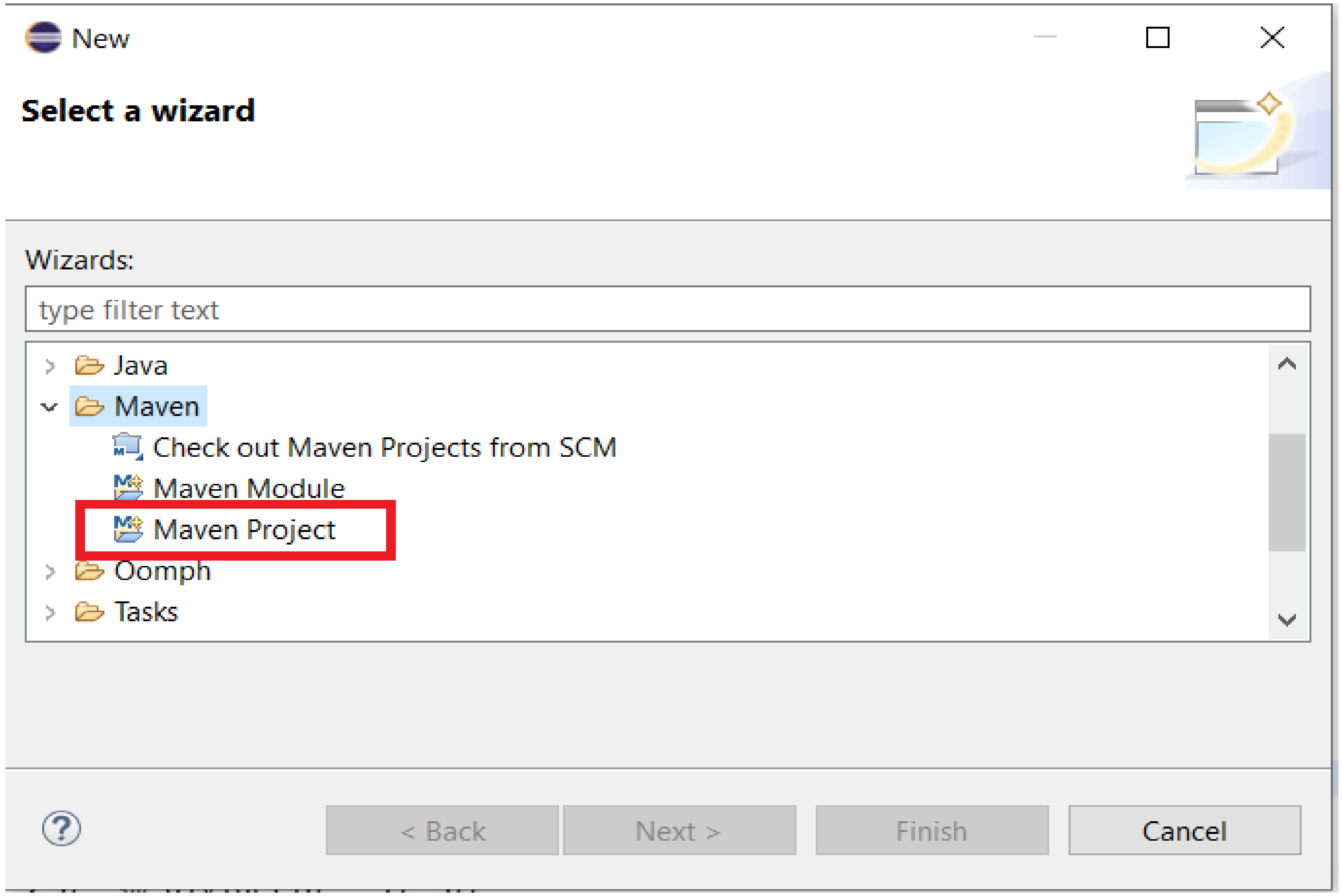
- Open Eclipse File → New → Project



```
ven.compiler.target>1.7</m
perties>
dencies>
pendency>
 <groupId>com.oracle</grou
```

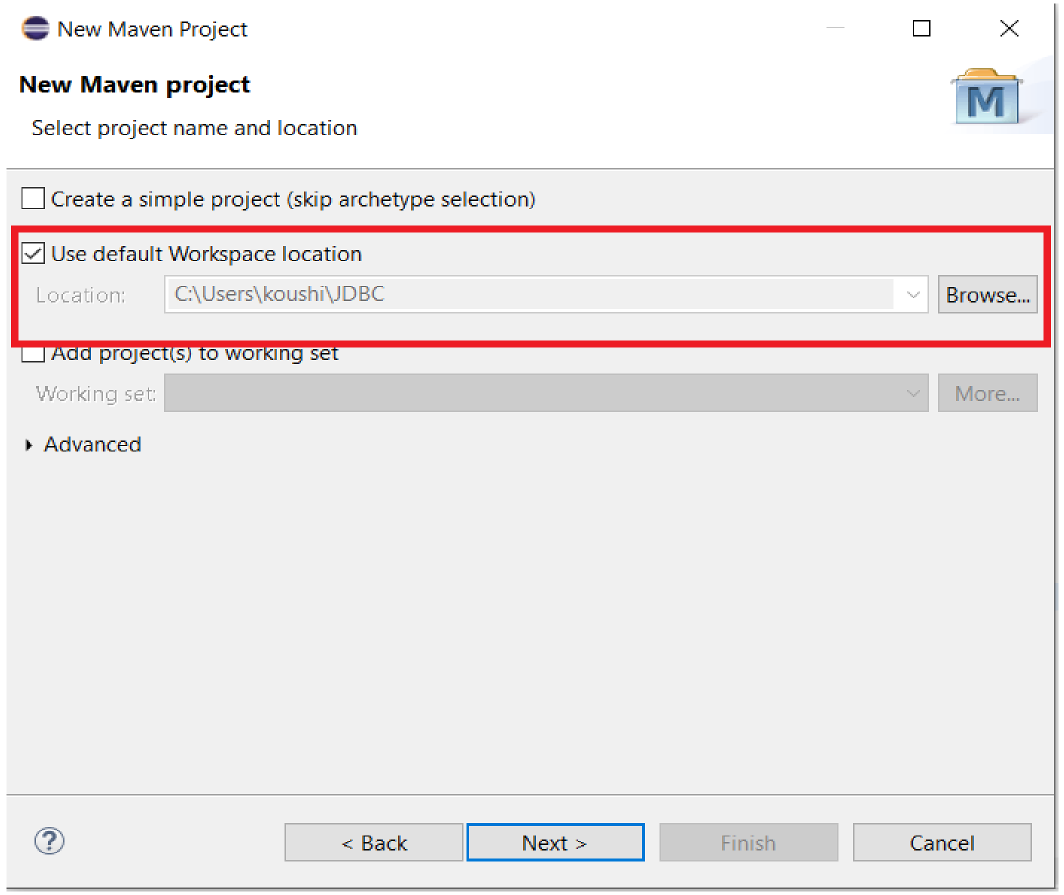
Configuring a JDBC development environment (Using Maven)

- **Select “Maven Project” and Click “Next”**



Configuring a JDBC development environment (Using Maven)

- **Choose the workspace and Click “Next”**


New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☑ Use default Workspace location

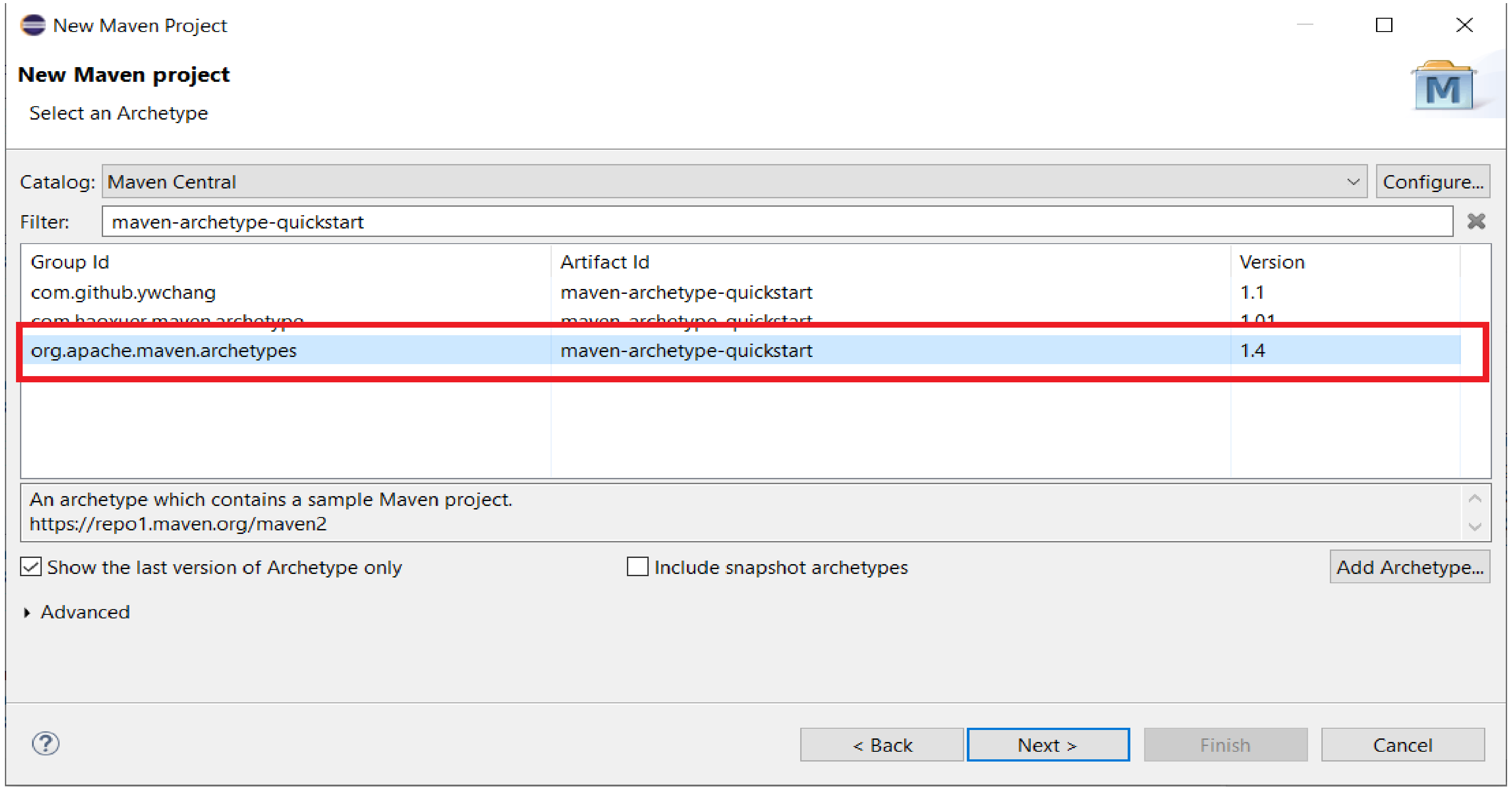
Location: C:\Users\koushi\JDBC

☐ Add project(s) to working set

Working set:

Configuring a JDBC development environment (Using Maven)

- **Choose the archtype and Click "Next"**



Catalog: Maven Central
Filter: maven-archetype-quickstart

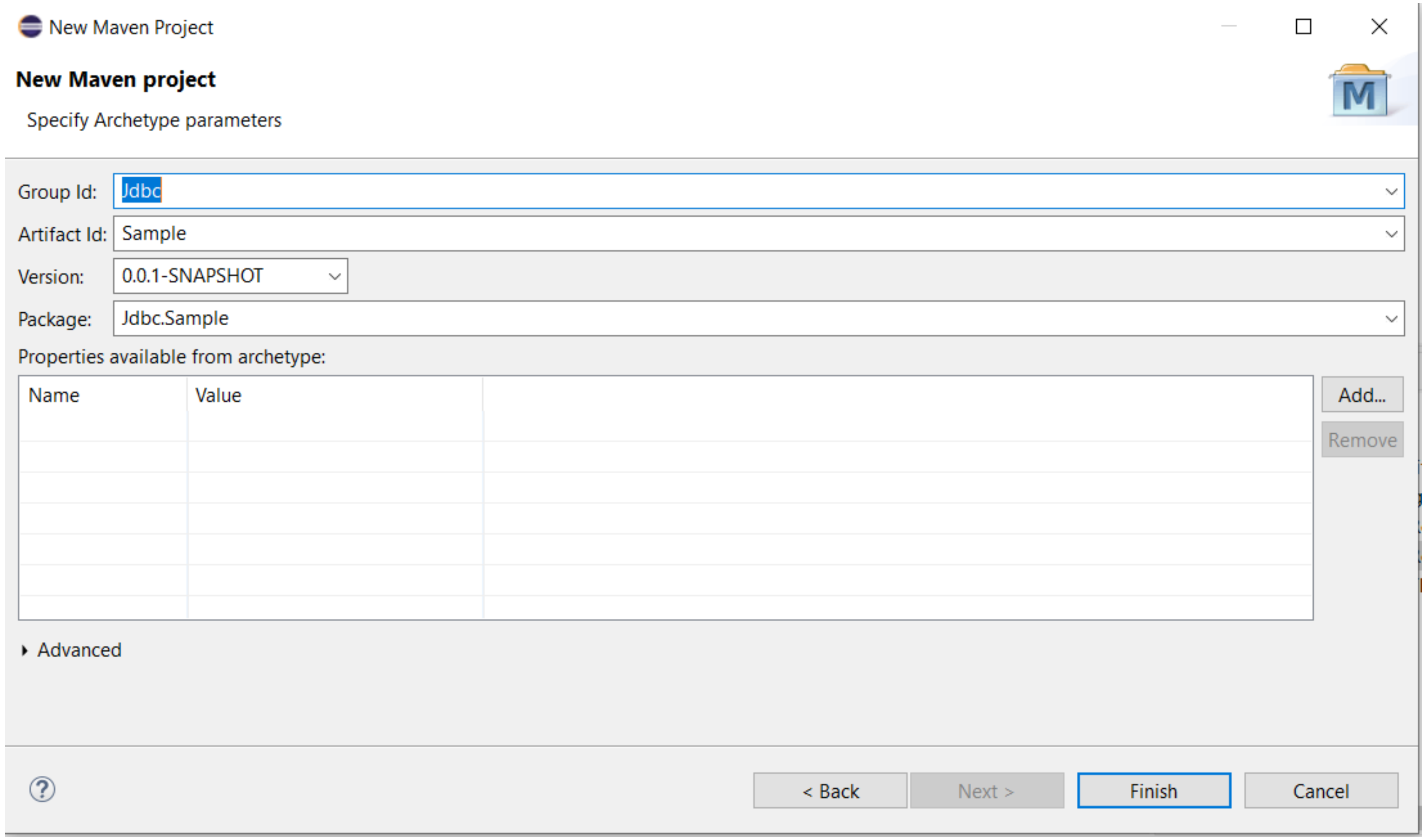
Group Id	Artifact Id	Version
com.github.ywchang	maven-archetype-quickstart	1.1
com.haoxuer.maven.archetype	maven-archetype-quickstart	1.01
org.apache.maven.archetypes	maven-archetype-quickstart	1.4

An archetype which contains a sample Maven project.
https://repo1.maven.org/maven2

[✓] Show the last version of Archetype only
[ ] Include snapshot archetypes

Configuring a JDBC development environment (Using Maven)

- **Enter Group ID and Artifact id …Package will be named automatically. Click Finish**

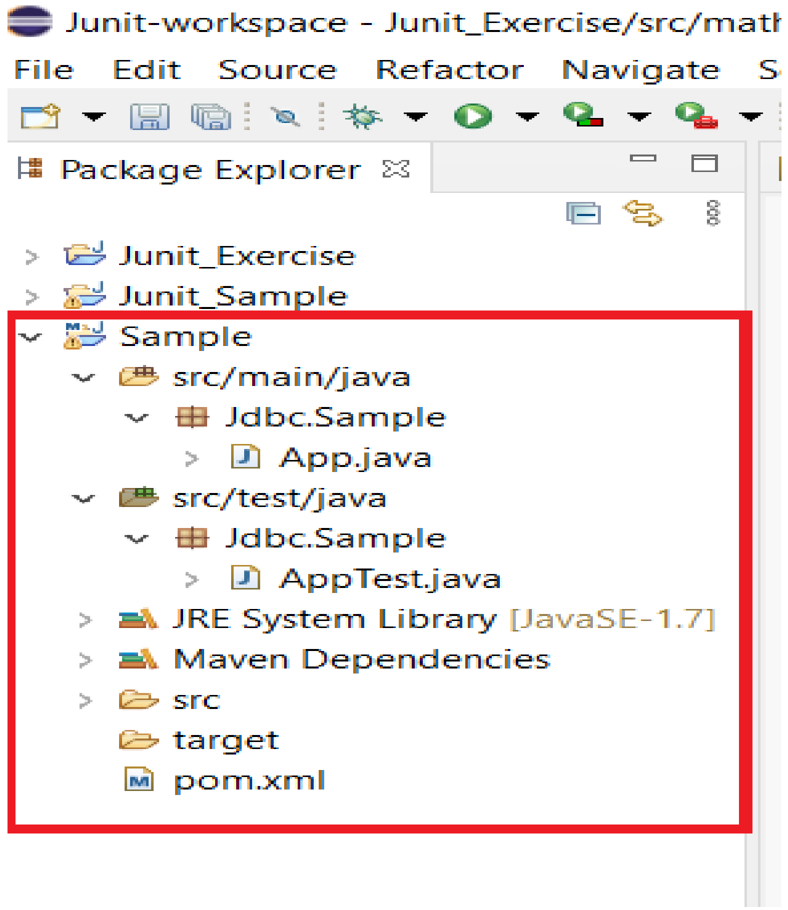


Group Id: Jdbc
Artifact Id: Sample
Version: 0.0.1-SNAPSHOT
Package: Jdbc.Sample
Properties available from archetype:

Name	Value

Configuring a JDBC development environment (Using Maven)

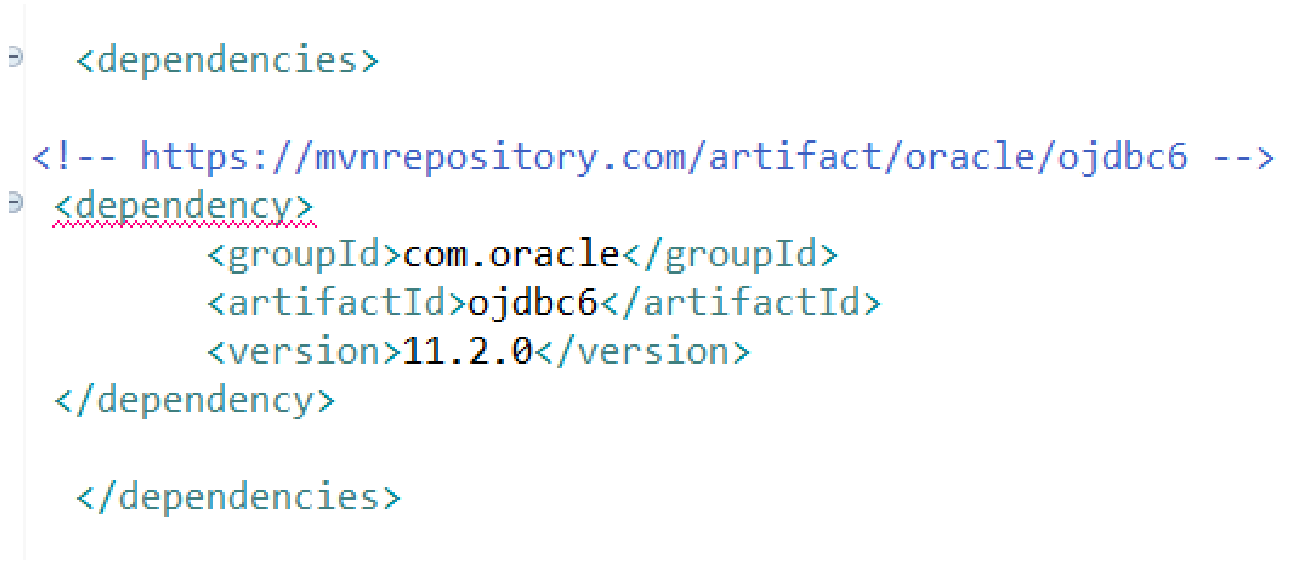
- **Maven project was built**


Junit-workspace - Junit_Exercise/src/matl
File Edit Source Refactor Navigate S
Package Explorer

```
> Junit_Exercise
> Junit_Sample
v Sample
  v src/main/java
    v Jdbc.Sample
      > App.java
  v src/test/java
    v Jdbc.Sample
      > AppTest.java
  > JRE System Library [JavaSE-1.7]
  > Maven Dependencies
  > src
    target
    pom.xml
```

Configuring a JDBC development environment (Using Maven)

- **Add the dependencies in “pom.xml”**

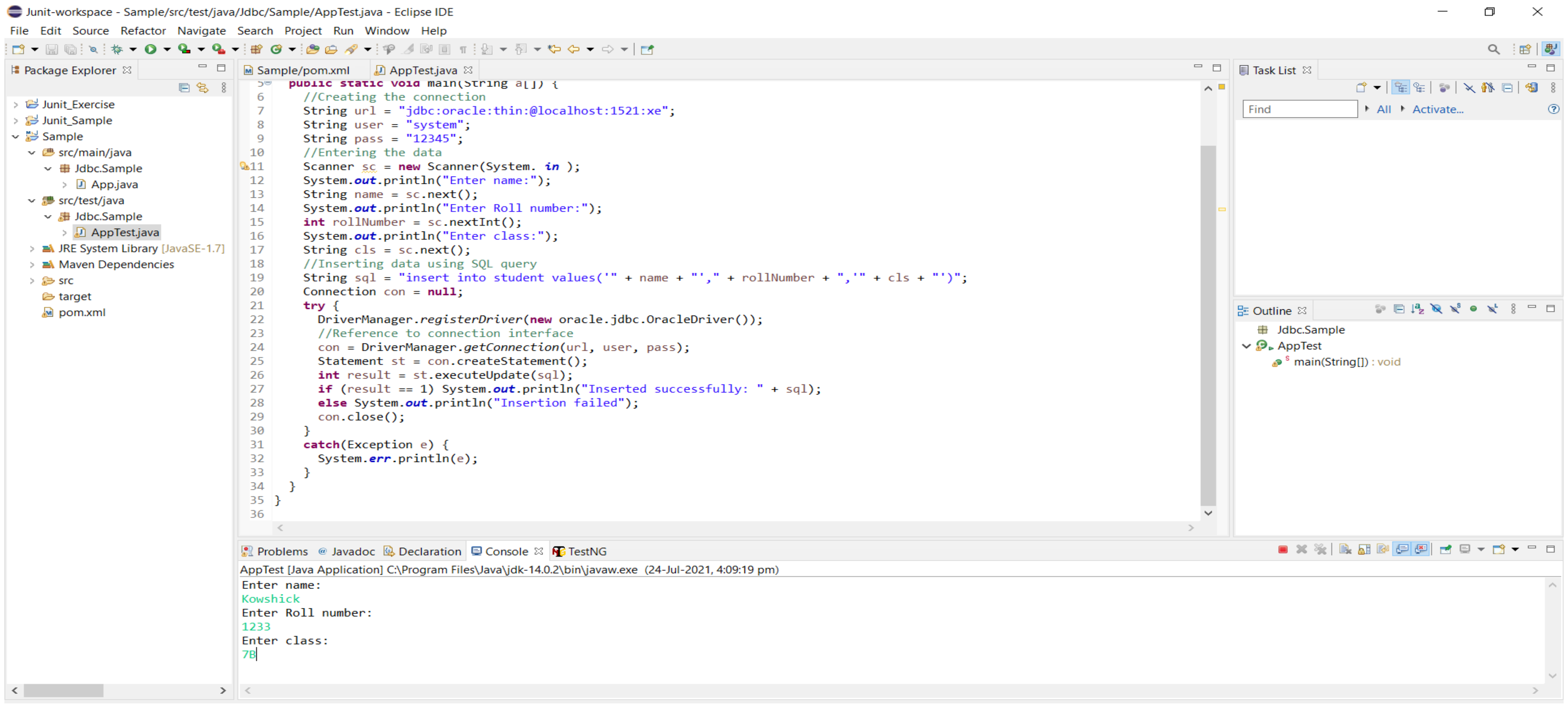


```
  <dependencies>

<!-- https://mvnrepository.com/artifact/oracle/ojdbc6 -->
 <dependency>
        <groupId>com.oracle</groupId>
        <artifactId>ojdbc6</artifactId>
        <version>11.2.0</version>
 </dependency>

  </dependencies>
```

Configuring a JDBC development environment (Using Maven)



```java
    public static void main(String a[]) {
      //Creating the connection
      String url = "jdbc:oracle:thin:@localhost:1521:xe";
      String user = "system";
      String pass = "12345";
      //Entering the data
      Scanner sc = new Scanner(System. in );
      System.out.println("Enter name:");
      String name = sc.next();
      System.out.println("Enter Roll number:");
      int rollNumber = sc.nextInt();
      System.out.println("Enter class:");
      String cls = sc.next();
      //Inserting data using SQL query
      String sql = "insert into student values('" + name + "','" + rollNumber + "','" + cls + "')";
      Connection con = null;
      try {
        DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
        //Reference to connection interface
        con = DriverManager.getConnection(url, user, pass);
        Statement st = con.createStatement();
        int result = st.executeUpdate(sql);
        if (result == 1) System.out.println("Inserted successfully: " + sql);
        else System.out.println("Insertion failed");
        con.close();
      }
      catch(Exception e) {
        System.err.println(e);
      }
    }
}
```

AppTest [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (24-Jul-2021, 4:09:19 pm)

```
Enter name:
Kowshick
Enter Roll number:
1233
Enter class:
7B
```

Example #1

Table creation

```
 CREATE TABLE  student (
Id int(11) NOT NULL,
Name varchar(45) NOT NULL,
Course varchar(45) NOT NULL,
Email varchar(45) NOT NULL,
PRIMARY KEY (Id));
```

Example #1 (Create table)

```
public class JdbcInsertDemo {

public static void main(String[] args) {
            String dbURL = "jdbc:oracle:thin:@localhost:1521:XE”;
               String username = "root";
               String password = "secret";

               try (Connection conn = DriverManager.getConnection(dbURL, username, password)) {

               Statement stmt = conn.createStatement();  {
                   String sql = "CREATE TABLE Student" +
                                           "(ID INTEGER not NULL, " +
                                           " NameVARCHAR(255), " +
                                           " Course VARCHAR(255), " +
                                           " Email VARCHAR(255), " +
                                           " PRIMARY KEY ( ID))";

                           stmt.executeUpdate(sql);
                          System.out.println("Created table in given database...");

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

Example #1 (Insert)

Inserting a new record into the table (101,”Bibin”,”Maths”,”bibin@gmail.com”)

```
public class JdbcInsertDemo {

public static void main(String[] args) {
            String dbURL = " jdbc:oracle:thin:@localhost:1521:XE”;
               String username = "root";
               String password = "secret";

               try (Connection conn = DriverManager.getConnection(dbURL, username, password)) {

               String sql = "INSERT INTO Users (Id, Name , Course , email) VALUES (?, ?, ?, ?)";
               PreparedStatement statement = conn.prepareStatement(sql);
               statement.setString(1, “101");
               statement.setString(2, “Bibin");
               statement.setString(3, “Maths");
               statement.setString(4, “bibin@gmail.com");
               int rowsInserted = statement.executeUpdate();
               if (rowsInserted > 0) {
                              System.out.println("A new user was inserted successfully!"); }

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

Example #1 (Select)

Displaying all the records from table

```
public class JdbcSelectDemo {

   public static void main(String[] args) {
                String dbURL = “jdbc:oracle:thin@localhost:1521:XE”;
                String username = "root";
                String password = "secret";
                try (Connection conn = DriverManager.getConnection(dbURL, username, password)) {
                            String sql = "SELECT * FROM School";
                            Statement statement = conn.createStatement();
                            ResultSet result = statement.executeQuery(sql);
                            int count = 0;
                        while (result.next()){
                              String id= result.getString(2);
                              String name= result.getString(3);
                              String course= result.getString(“Course");
                              String email = result.getString(“Email");
                              String output = "User #%d: %s - %s - %s - %s";
                              System.out.println(String.format(output, ++count,id, name, course, email));
                       }
                } catch (SQLException ex) {
                            ex.printStackTrace();
                }
} }
```

Example #1 (Update)

Updating the course to “Physics” for the name “Bibin”

```
public class JdbcUpdateDemo {

        public static void main(String[] args) {
        String dbURL = " jdbc:oracle:thin@localhost:1521:XE";
        String username = "root";
        String password = "secret";
      try (Connection conn = DriverManager.getConnection(dbURL, username, password)) {
      String sql = "UPDATE Users SET Course=?, email=? WHERE username=?";

        PreparedStatement statement = conn.prepareStatement(sql);
        statement.setString(1, “Physics");
        statement.setString(2, “bibin6420@gmail.com");
        statement.setString(3, "Bibin");

       int rowsUpdated = statement.executeUpdate();
        if (rowsUpdated > 0) {
                System.out.println("An existing user’s details was updated successfully!");
        }
     } catch (SQLException ex) {
        ex.printStackTrace();
        }
    }
}
```

Example #1 (Delete)

Deleting the record from table whose name is “Bibin”

```
public class JdbcDeleteDemo {

        public static void main(String[] args) {
        String dbURL = " jdbc:oracle:thin:@localhost:1521:XE";
        String username = "root";
        String password = "secret";

        try (Connection conn = DriverManager.getConnection(dbURL, username, password)) {

                String sql = "DELETE FROM Users WHERE username=?";
                PreparedStatement statement = conn.prepareStatement(sql);
                statement.setString(1, “Bibin");
                int rowsDeleted = statement.executeUpdate();
                if (rowsDeleted > 0) {
                        System.out.println("A user was deleted successfully!");
                }

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

Example #2

Table creation

```
create table employee(E_Id number(10),Name varchar2(40),Dept varchar2(40));
```

Example #2 (Stored Procedure)

```
import java.sql.*;
class Sample  {
public static void main(String args[]) {
try{

//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
  Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from employee");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object
con.close();
}catch(Exception e){ System.out.println(e);}
}  }
```

Example #2 (Stored Procedure)

```
import java.math.BigDecimal;
import java.sql.*;

public class StoreProcedureOutParameter {
    public static void main(String[] args) {
        String createSP = "CREATE OR REPLACE PROCEDURE get_employee_by_id( "
                + " p_id IN EMPLOYEE.ID%TYPE, "
                + " o_name OUT EMPLOYEE.NAME%TYPE, "
                + " o_salary OUT EMPLOYEE.SALARY%TYPE, "
                + " o_date OUT EMPLOYEE.CREATED_DATE%TYPE) "
                + " AS "
                + " BEGIN "
                + "     SELECT NAME, SALARY, CREATED_DATE INTO o_name, o_salary, o_date from EMPLOYEE WHERE ID = p_id; "
                + " END;";
```

Example #2 (Stored Procedure)

```
String runSP = "{ call get_employee_by_id(?,?,?,?) }";
        try (Connection conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe", "system", "Password123");
             Statement statement = conn.createStatement();
             CallableStatement callableStatement = conn.prepareCall(runSP)) {

            // create or replace stored procedure
            statement.execute(createSP);
            callableStatement.setInt(1, 3);
            callableStatement.registerOutParameter(2, java.sql.Types.VARCHAR);
            callableStatement.registerOutParameter(3, Types.DECIMAL);
            callableStatement.registerOutParameter(4, java.sql.Types.DATE);

            // run it
            callableStatement.executeUpdate();
```

Example #2 (Stored Procedure)

```
String name = callableStatement.getString(2);
            BigDecimal salary = callableStatement.getBigDecimal(3);
            Timestamp createdDate = callableStatement.getTimestamp(4);

            System.out.println("name: " + name);
            System.out.println("salary: " + salary);
            System.out.println("createdDate: " + createdDate);

        } catch (SQLException e) {
            System.err.format("SQL State: %s\n%s", e.getSQLState(), e.getMessage());
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }    }
}
```

Quiz

1. Select the packages in which JDBC classes are defined?

a) java.sql and javax.sql

b) rdb and javax.rdb

c) jdbc and java.jdbc.sql

d) jdbc and javax.jdbc

a) java.sql and javax.sql

Quiz

2. Which of the following method is used to perform DML statements in JDBC?

a) executeResult()

b) execute()

c) executeQuery()

d) executeUpdate()

d) executeUpdate()

Quiz

3. Which of the following is not a valid statement in JDBC?

a) Statement

b) QueryStatement

c) PreparedStatement

d) CallableStatement

b) QueryStatement

Quiz

4. Which is used to call the stored procedures and functions?

a) CallableStatement Interface

b) PreparedStatement Interface

c) Both

d) None

a) CallableStatement Interface

Quiz

5. The ResultSet.next method is used to move to the next row of the ResultSet, making it the current row.

a) True

b) False

a) True

THANK YOU

SmartCliff
Career Mobility Solutions