JDBC

The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a relational database.

JDBC helps you to write Java applications that manage these three programming activities:

1. *Connect to a data source, like a database*
2. *Send queries and update statements to the database*
3. *Retrieve and process the results received from the database in answer to your query*

```java
import java.sql.*;
class JDBCDemo {

    public static void main(String[] args)throws Exception {
        Connection con = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1527:XE",
                "rahul",
                "cdac");// it made conncection between java application and data
bse

        Statement stmt = con.createStatement();//statement object creation for r
equirement or send sql query
        ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");//result
set get from the databse after the operation

        //it can it iterate thoght the result set
        while (rs.next()) {
            int x = rs.getInt("a");
            String s = rs.getString("b");
            float f = rs.getFloat("c");
        }
    }
}

```

// JDBC Product Components

// JDBC includes four components:

// The JDBC API — The JDBC™ API provides programmatic access to relational data from the Java™ programming language. Using the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source. The JDBC API can also interact with multiple data sources in a distributed, heterogeneous environment.

// The JDBC API is part of the Java platform, which includes the Java™ Standard Edition (Java™ SE) and the Java™ Enterprise Edition (Java™ EE). The JDBC 4.0 API is divided into two packages: java.sql and javax.sql. Both packages are included in the Java SE and Java EE platforms.

// JDBC Driver Manager — The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager has traditionally been the backbone of the JDBC architecture. It is quite small and simple.

// The Standard Extension packages javax.naming and javax.sql let you use a DataSource object registered with a Java Naming and Directory Interface™ (JNDI) naming service to establish a connection with a data source. You can use either connecting mechanism, but using a DataSource object is recommended whenever possible.

// JDBC Test Suite — The JDBC driver test suite helps you to determine that JDBC drivers will run your program. These tests are not comprehensive or exhaustive, but they do exercise many of the important features in the JDBC API.

// JDBC-ODBC Bridge — The Java Software bridge provides JDBC access via ODBC drivers. Note that you need to load ODBC binary code onto each client

machine that uses this driver. As a result, the ODBC driver is most appropriate on a corporate network where client installations are not a major problem, or for application server code written in Java in a three-tier architecture.

// This Trail uses the first two of these four JDBC components to connect to a database and then build a java program that uses SQL commands to communicate with a test relational database. The last two components are used in specialized environments to test web applications, or to communicate with ODBC-aware DBMSs.

Driver software: which convert java calls to database calls and vice versa

Connectivity: connection between java application and database

Statement object: to send some requirement (SQL Query) and get response from database in a result set

JDBC:--- is a technology to communicate with database java applicaton

JDBC Interview Questions:-

----

1. Explain what do you mean by JDBC?

2. What are the steps to connect to a database in java?

3. What are the JDBC API components?

4. How to you load the drivers in JDBC?

5. What is the purpose of JDBC ResultSet interface?

6. Explain how you can establish a connection ?

7. What are the different types of statements in JDBC ?

8. Have you used prepared statements? Where have you used prepared statements ?

9. How do you create JDBC statements ?

10. What is the difference between a Statement and a PreparedStatement?

11. What are callable statements ?

12. What are types of JDBC drivers?

13. How do you retrieve data from a result set, explain with an examplet?

14. Explain about stored procedure ?

15. When do we look for batch updates ?

16. What is the return type of Class.forName() method?

17. How can we execute stored procedures using CallableStatement?

18. What is the role of the JDBC DriverManager class?

19. What are the functions of the JDBC Connection interface?

20. What is the major difference between java.util.Date and java.sql.Date data type?

1. What do you mean by JDBC?

JDBC (Java Database Connectivity) is an API in Java that allows Java programs to connect to and interact with databases.

✅ **Example**: You can use JDBC to run SQL queries from a Java program.

---

2. Steps to connect to a database in Java?

1. Load the driver
2. Create a connection
3. Create a statement
4. Execute the query
5. Process the result
6. Close the connection

✅ **Example**:

java

CopyEdit

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

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "user",
"password");

Statement stmt = conn.createStatement();

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

while(rs.next()) {

   System.out.println(rs.getString("name"));

}

conn.close();
```

---

3. JDBC API Components

- **DriverManager**
- **Connection**
- **Statement**
- **PreparedStatement**
- **CallableStatement**
- **ResultSet**

---

4. How do you load the drivers in JDBC?

Using Class.forName() method.

✅ **Example**:

java

CopyEdit

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

---

5. Purpose of JDBC ResultSet interface?

To hold and process the result returned by SQL queries (SELECT).

✅ **Example**:

java

CopyEdit

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

while (rs.next()) {
```

```
    System.out.println(rs.getInt("id") + " " + rs.getString("name"));
}
```

---

6. How can you establish a connection?

By using DriverManager.getConnection().

✅ **Example**:

java

CopyEdit

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "root", "password");
```

---

7. Different types of statements in JDBC

- Statement
- PreparedStatement
- CallableStatement

---

8. Have you used PreparedStatement? Where?

Yes, used it to avoid SQL injection and for dynamic SQL.

✅ **Example**:

java

CopyEdit

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE email = ?");
ps.setString(1, "abc@example.com");
ResultSet rs = ps.executeQuery();
```

---

9. How do you create JDBC statements?

Using Connection object.

✅ **Example**:

java

CopyEdit

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

---

10. Difference between Statement and PreparedStatement

Statement	PreparedStatement
Simple SQL	Precompiled SQL
Slower for repeated use	Faster for repeated use
Prone to SQL injection	Safe from SQL injection

---

11. What are CallableStatements?

Used to execute **stored procedures** from Java.

✅ **Example**:

java

CopyEdit

```
CallableStatement cs = conn.prepareCall("{call getEmployee(?, ?)}");
cs.setInt(1, 101);
cs.registerOutParameter(2, Types.VARCHAR);
cs.execute();
```

---

12. Types of JDBC drivers

1. **JDBC-ODBC bridge driver**
2. **Native-API driver**
3. **Network Protocol driver**
4. **Thin driver (Pure Java)** ✅ Most common

---

13. How to retrieve data from ResultSet?

✅ **Example**:

java

CopyEdit

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
while(rs.next()) {
   System.out.println(rs.getInt("id") + ", " + rs.getString("name"));
```

}

---

14. What is a stored procedure?

A stored procedure is a group of SQL statements stored in the database that can be called by name.

✅ **Example** (MySQL):

sql

CopyEdit

CREATE PROCEDURE getUsers()

BEGIN

SELECT * FROM users;

END

---

15. When do we use batch updates?

When we need to execute **multiple SQL statements together**, like inserting many rows.

✅ **Example**:

java

CopyEdit

Statement stmt = conn.createStatement();

stmt.addBatch("INSERT INTO students VALUES (1, 'John')");

stmt.addBatch("INSERT INTO students VALUES (2, 'Jane')");

stmt.executeBatch();

---

16. Return type of Class.forName()

It returns a **Class<?>** object.
It's used to load the class into memory (side-effect: driver gets registered).

---

17. How to execute stored procedures using CallableStatement?

✅ **Example**:

java

CopyEdit

CallableStatement cs = conn.prepareCall("{call getStudentName(?)}");

cs.setInt(1, 101);

ResultSet rs = cs.executeQuery();

---

18. Role of DriverManager class

Manages the list of database drivers and **establishes connections**.

✅ **Example**:

java

CopyEdit

Connection conn = DriverManager.getConnection(...);

---

19. Functions of JDBC Connection interface

- Create Statement
- Manage transactions
- Get metadata
- Close the connection

✅ **Example**:

java

CopyEdit

conn.setAutoCommit(false);

conn.commit();

conn.rollback();

---

20. Difference: java.util.Date vs java.sql.Date

- java.util.Date: General-purpose date/time
- java.sql.Date: Only contains date (no time), used for DB

✅ **Example**:

java

CopyEdit

java.sql.Date sqlDate = new java.sql.Date(new java.util.Date().getTime());

```java
package com.rahul.DBUtil;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;


public class DBUtils {

    private static Connection connection;
    private static final String DB_URL="jdbc:mysql://localhost:3306/javaweb";
    private static final String USER_NAME="rahul";
    private static final String PASSWORD="cdac";


    //open connection to DB connection

    public static void openConnection()throws Exception {
        //1 load jdbc driver (optional step)
//      Class.forname("com.mysql.cj.jdbc.Driver")

        //2 get fixed data connection from driverManager
        //public static Connection getConnection(String url,String uername,
        //String password)throws SQLException

        //singleton instance of the DB connection
        //eager singleton is the by static block
        /*
         * static{
         * connection=DriverManager.getConnection(DB_URL, USER_NAME, PASSWORD);
         * }
         */
        //lazy singleton
        if(connection==null) {
            connection=DriverManager.getConnection(DB_URL, USER_NAME, PASSWORD);
        }
    }
    public static Connection getConnection() {
        return connection;
    }
    public static void closeConnection() throws SQLException {
        if(connection!=null) {
            connection.close();
            connection=null;
        }
    }
}

```

```java
package com.rahul.tester;

import java.sql.Connection;

import static com.rahul.DBUtil.DBUtils.*;

//import com.rahul.DBUtil.*;

public class TestConnection {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {
//          //open
//          DBUtils.openConnection();
//          //get
//          Connection  conn=DBUtils.getConnection();
//          //close
//          DBUtils.closeConnection();

            //open
            openConnection();
            //get
            Connection conn=getConnection();
            System.out.println("Connected to Db..."+conn);
            //close
            closeConnection();
            System.out.println("Close connection");
        }catch(Exception e) {
            e.printStackTrace();
        }

        System.out.println("Main over");

    }

}
```