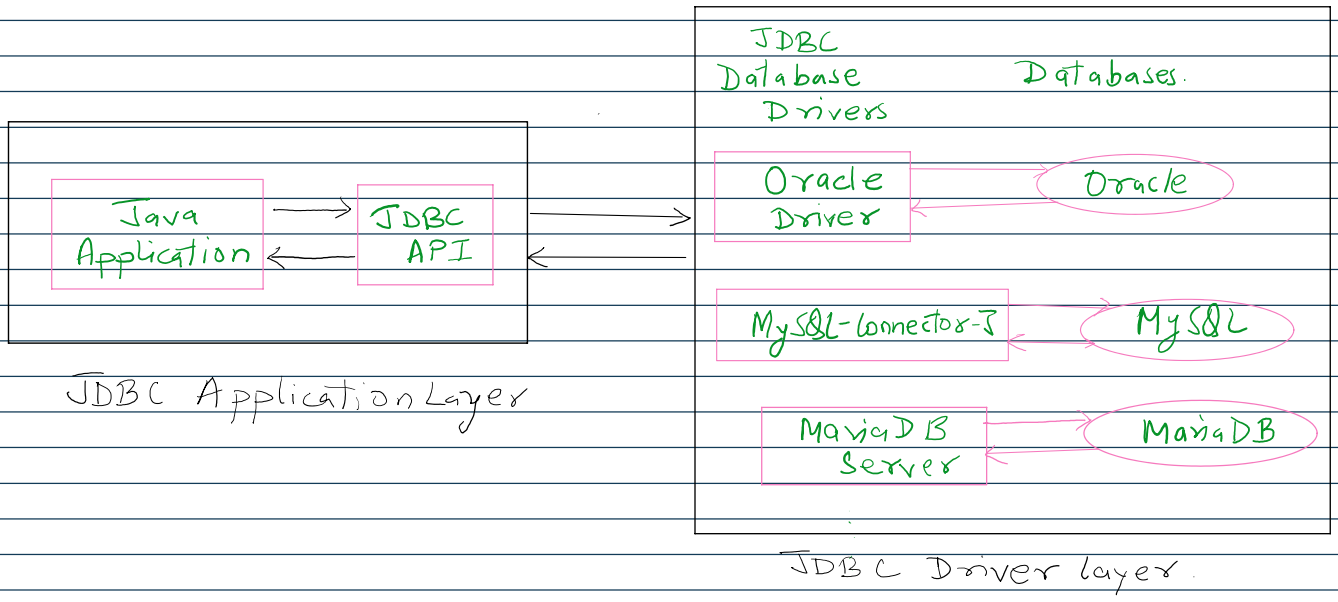
Introduction JDBC

15 April 2025 17:20

JDBC is a standard API for connecting and executing queries with databases. It is part of Java Standard Edition platform and is widely used in enterprise application.



JDBC Architecture

JDBC Architecture Overview

The JDBC API consist of two layers:

- Application Layer: Java application using JDBC API
- JDBC Driver Layer: Interfaces with specific databases(MySQL, Oracle, etc.)

There are 4 types of JDBC drivers:

Type-1: JDBC-ODBC bridge
Type-2: Native-API driver
Type-3: Network protocol driver
Type-4: Thin driver(mostly used driver)

JDBC API

Important Classes and Interfaces

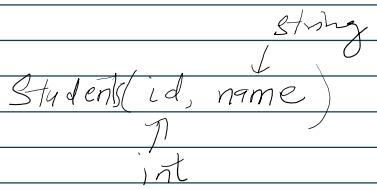
java.sql.DriverManager
java.sql.Connection(I)
java.sql.Statement(I)
java.sql.PreparedStatement(I)
java.sql.CallableStatement(I)
java.sql.ResultSet(I)
java.sql.ResultSetMetaData(I)
java.sql.DatabaseMetaData(I)
java.sql.SQLException

JDBC Workflow

1. **Import JDBC packages**

```
import java.sql.*;
```

```
2. Load and Register the Driver
   Class.forName("com.mysql.cj.jdbc.Driver"); //MySQL

   Class.forName("oracle.jdbc.OracleDriver"); //Oracle

3. Establish a Database connection
   Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/database_name", "root",
   "password");

                Or
   Connection con = DriverManager.getConnection(url, user, password);

4. Create a Statement
   Statement stmt = con.createStatement();

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

6. Process Results
      while(rs.next()){
          System.out.print("\nID:" + rs.getInt("id")+", Name: " + rs.getString("name"));
      }

7. Close the connection
   rs.close();
   stmt.close();
   con.close();
```



Using PreparedStatement(Parameterized Queries)
Helps prevent SQL injection and optimize performance.

```
String query = "SELECT * FROM students WHERE id = ?";
PreparedStatement ps = con.prepareStatement(query);
ps.setInt(1, 101);
ResultSet rs = ps.executeQuery();
while(rs.next()){
    System.out.print("\nName: " + rs.getString("name"));
}
```

Inserting Data Example

```
String query = "INSERT INTO students(id, name) VALUES(?,?)";
PreparedStatement ps = con.prepareStatement(query);
ps.setInt(1, 101);
Ps.setString(2, "Shiv");
int result = ps.executeUpdate();
System.out.print("\n"+result + " row(s) inserted.");
```

Updating and deleting data
Update:

```
String query = "UPDATE students SET name=? WHERE id=?";
PreparedStatement ps = con.prepareStatement(query);
ps.setString(1, "John Smith");
ps.setInt(2, 104);
ps.executeUpdate();
```

Delete:

```
String query = "DELETE students WHERE id=?";
PreparedStatement ps = con.prepareStatement(query);
ps.setInt(1, 104);
ps.executeUpdate();
```

Handling Exceptions

Always use try-catch blocks to handle exceptions gracefully.

```
try{
   //JDBC logic here
}catch(SQLException e){
   System.out.println("Database error: " + e.getMessage());
}
```