Java JDBC with MySQL - Complete Notes & CRUD Project

# 1. Introduction to Java JDBC

JDBC (Java Database Connectivity) is a Java API for connecting and executing queries on databases.  
It provides a standard way to interact with relational databases like MySQL, Oracle, PostgreSQL.  
JDBC acts as a bridge between the Java application and the database.

# 2. Why JDBC?

Without JDBC, Java applications cannot talk directly to a database.  
JDBC simplifies:  
- Connection to DB  
- Execution of SQL queries  
- Retrieval and processing of results

# 3. Data Types in JDBC

JDBC maps SQL data types to Java data types for smooth data handling.  
  
SQL Type | Java Equivalent | Usage Example  
----------------------------------------------------------------------------------------------------------------------  
INT | int / Integer | Student ID, Age  
VARCHAR(n) | String | Name, Email  
DATE | java.sql.Date | Date of enrollment  
DECIMAL, FLOAT | float / double | Fees, Marks  
BOOLEAN | boolean | Yes/No flags  
BLOB | java.sql.Blob | Images, Documents  
CLOB | java.sql.Clob | Large text files

# 4. JDBC Architecture

The architecture has four main components:  
1. JDBC API - Classes and interfaces (Connection, Statement, ResultSet)  
2. JDBC Driver Manager - Manages database drivers  
3. JDBC Drivers - Vendor-specific implementation for MySQL, Oracle, etc.  
4. Database - Actual RDBMS (e.g., MySQL server)  
  
Flow: Java App → JDBC API → JDBC Driver Manager → JDBC Driver → Database

# 5. MySQL Setup and Commands

### Basic Commands ###  
1. Login to MySQL:  
 mysql -u root -p  
  
2. Show all Databases:  
 SHOW DATABASES;  
  
3. Create Database:  
 CREATE DATABASE university;  
  
4. Select Database:  
 USE university;  
  
5. Show Tables:  
 SHOW TABLES;  
  
6. Create Table:  
 CREATE TABLE engineeringstudents (  
 student\_id INT PRIMARY KEY,  
 department VARCHAR(25),  
 first\_name VARCHAR(25),  
 last\_name VARCHAR(25),  
 passedyear INT,  
 universityrank INT  
 );  
  
7. Insert Data:  
 INSERT INTO engineeringstudents VALUES (101, 'Computer Science', 'Siddharth', 'Sharma', 2023, 5);  
  
8. View Data:  
 SELECT \* FROM engineeringstudents;  
  
9. Update Data:  
 UPDATE engineeringstudents SET universityrank = 1 WHERE student\_id = 101;  
  
10. Delete Data:  
 DELETE FROM engineeringstudents WHERE student\_id = 101;  
  
11. Drop Table:  
 DROP TABLE engineeringstudents;  
  
12. Drop Database:  
 DROP DATABASE university;

# 6. Java Setup

1. Install JDK: https://www.oracle.com/java/technologies/javase-downloads.html  
2. Install IDE: IntelliJ IDEA / Eclipse  
3. Add MySQL JDBC Driver: Download mysql-connector-j.jar and add to project classpath  
4. JDBC Connection URL: jdbc:mysql://localhost:3306/university

# 7. JDBC Implementation Steps

1. Load JDBC Driver: Class.forName("com.mysql.cj.jdbc.Driver");  
2. Establish Connection: DriverManager.getConnection(url, user, password);  
3. Create Statement: con.createStatement();  
4. Execute Query: stmt.executeQuery("SELECT \* FROM engineeringstudents");  
5. Process Results: rs.next();  
6. Close Connections: rs.close(); stmt.close(); con.close();

# 8. Full CRUD Java Code

Sample Code for Starting JDBC and My SQL:

**package** jdbc;

**import** java.sql.SQLException;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**public** **class** jdbcProject{

**public** **static** **void** main(String [] args) **throws** SQLException {

String url = "jdbc:mysql://localhost:3306/university";

String user ="root";

String password = "password";

String query ="SELECT \* FROM engineeringstudents";

**try** {

// Load MySQL JDBC Driver (optional for newer versions)

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

System.***out***.println("Connected to MySQL database!");

}**catch** (ClassNotFoundException e) {

System.***out***.println("MySQL JDBC Driver not found.");

e.printStackTrace();

}

**try**{

// Establish connection

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

// Create statement and execute query

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(query);

**while**(rs.next()){

String UniversityData="";

**for**(**int** i = 1; i<=6; i++){

UniversityData += rs.getString(i) + ":";

}

System.***out***.println(UniversityData);

}

// Close resources

rs.close();

stmt.close();

con.close();

}**catch** (SQLException e) {

System.***out***.println("SQL error occurred.");

e.printStackTrace();

}

}

}

Complete CRUD code as provided earlier for Adding, Viewing, Updating, and Deleting students.

Got it! I’ll go through your JdbcCRUD code line by line, add **detailed comments** in the code itself, and then provide a **separate notes section** explaining why each major part is used and what it does.

Here’s the fully commented version of your code:

package jdbc; // Package name for organizing classes

import java.sql.\*; // Import JDBC classes for database connectivity

import java.util.Scanner; // Import Scanner for user input

public class JdbcCRUD {

// Database connection details (URL, username, password)

static final String URL = "jdbc:mysql://localhost:3306/university";

static final String USER = "root";

static final String PASSWORD = "password";

public static void main(String[] args) {

try {

// 1. Load MySQL JDBC Driver - ensures JDBC knows about the MySQL driver

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

// 2. Establish connection to MySQL database using URL, USER, PASSWORD

Connection con = DriverManager.getConnection(URL, USER, PASSWORD);

// 3. Scanner object for user input

Scanner sc = new Scanner(System.in);

// 4. Infinite loop for menu until user chooses Exit

while (true) {

System.out.println("1. Add Student 2. View Students 3. Update Student 4. Delete Student 5. Exit");

int choice = sc.nextInt(); // Read user's choice

switch (choice) {

case 1: createStudent(con, sc); break; // Add student

case 2: readStudents(con); break; // View all students

case 3: updateStudent(con, sc); break; // Update student's rank

case 4: deleteStudent(con, sc); break; // Delete student

case 5:

con.close(); // Close DB connection before exit

sc.close(); // Close scanner

return; // Exit the program

default: System.out.println("Invalid Choice!"); // Wrong input

}

}

} catch (Exception e) {

e.printStackTrace(); // Print error details if something goes wrong

}

}

// Method to create (INSERT) a new student into DB

public static void createStudent(Connection con, Scanner sc) throws SQLException {

// Taking input from user

System.out.print("Enter ID: "); int id = sc.nextInt(); sc.nextLine(); // Consume newline

System.out.print("Enter Department: "); String dept = sc.nextLine();

System.out.print("Enter First Name: "); String fname = sc.nextLine();

System.out.print("Enter Last Name: "); String lname = sc.nextLine();

System.out.print("Enter Passed Year: "); int year = sc.nextInt();

System.out.print("Enter University Rank: "); int rank = sc.nextInt();

// SQL query to insert values into the table

String query = "INSERT INTO engineeringstudents VALUES (?, ?, ?, ?, ?, ?)";

// PreparedStatement to prevent SQL injection and set values dynamically

PreparedStatement ps = con.prepareStatement(query);

ps.setInt(1, id);

ps.setString(2, dept);

ps.setString(3, fname);

ps.setString(4, lname);

ps.setInt(5, year);

ps.setInt(6, rank);

// Execute the INSERT query

ps.executeUpdate();

System.out.println("Student Added Successfully!");

}

// Method to READ all students from DB

public static void readStudents(Connection con) throws SQLException {

Statement stmt = con.createStatement(); // Create simple statement

ResultSet rs = stmt.executeQuery("SELECT \* FROM engineeringstudents"); // Execute SELECT query

// Iterate over result set and print each record

while (rs.next()) {

System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)+" "+rs.getString(4)+" "+rs.getInt(5)+" "+rs.getInt(6));

}

}

// Method to UPDATE student's rank based on student\_id

public static void updateStudent(Connection con, Scanner sc) throws SQLException {

System.out.print("Enter Student ID to Update: "); int id = sc.nextInt();

System.out.print("Enter New University Rank: "); int rank = sc.nextInt();

// Update query with placeholders

String query = "UPDATE engineeringstudents SET universityrank = ? WHERE student\_id = ?";

PreparedStatement ps = con.prepareStatement(query);

ps.setInt(1, rank); // Set new rank

ps.setInt(2, id); // Set student ID

// Execute the UPDATE query

int rows = ps.executeUpdate();

if (rows > 0) System.out.println("Student Updated Successfully!");

else System.out.println("Student ID Not Found!");

}

// Method to DELETE a student record

public static void deleteStudent(Connection con, Scanner sc) throws SQLException {

System.out.print("Enter Student ID to Delete: "); int id = sc.nextInt();

// Delete query with placeholder

String query = "DELETE FROM engineeringstudents WHERE student\_id = ?";

PreparedStatement ps = con.prepareStatement(query);

ps.setInt(1, id); // Set student ID

// Execute the DELETE query

int rows = ps.executeUpdate();

if (rows > 0) System.out.println("Student Deleted Successfully!");

else System.out.println("Student ID Not Found!");

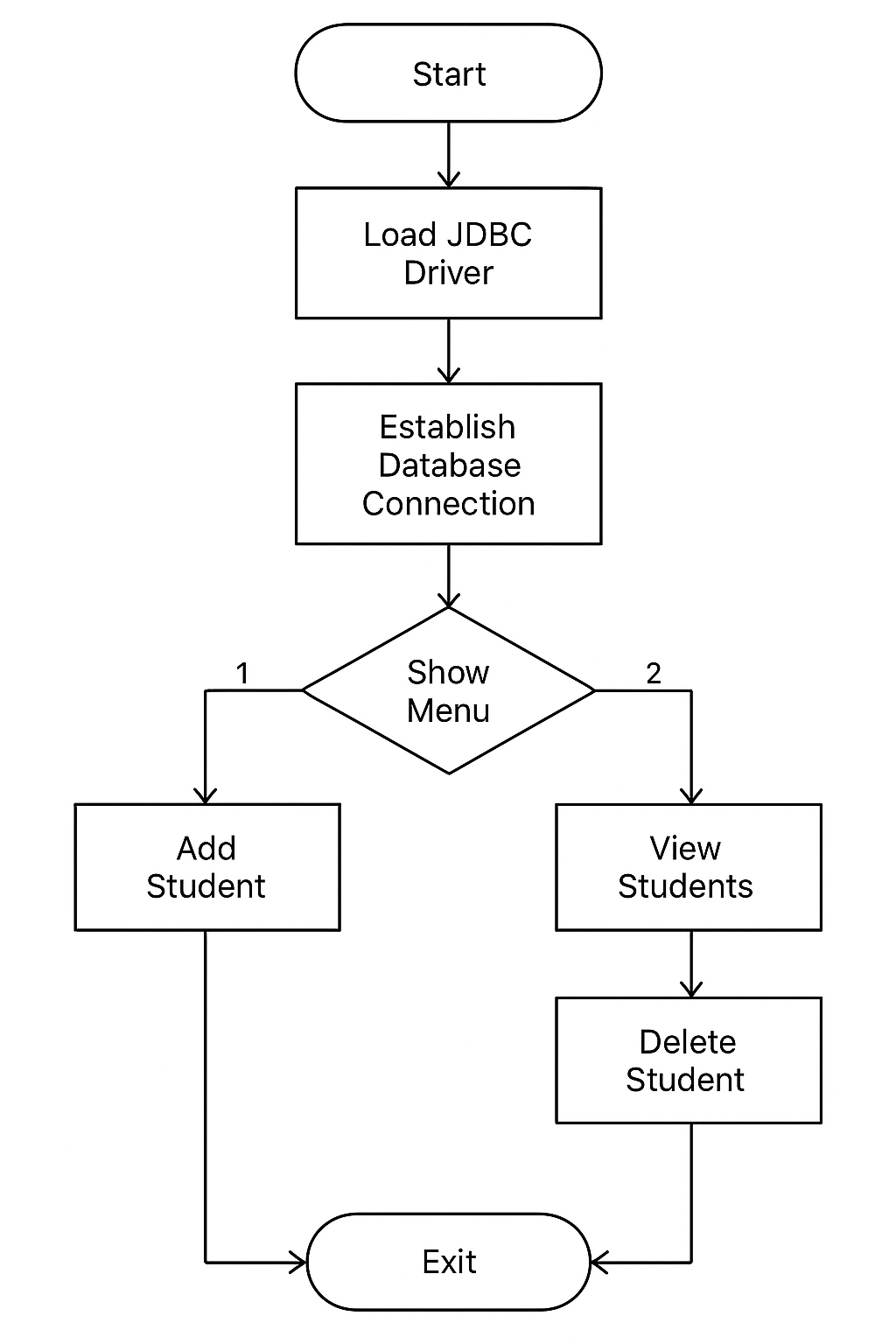
}

}

**Notes for Each Section:**

1. **Class.forName("com.mysql.cj.jdbc.Driver")**
   * Loads MySQL JDBC driver into memory so Java can talk to MySQL.
2. **DriverManager.getConnection()**
   * Creates a connection object to interact with the database.
3. **Scanner for Input**
   * Allows reading user input from the console.
4. **PreparedStatement**
   * Used for INSERT, UPDATE, DELETE queries.
   * Prevents SQL injection by using placeholders ? instead of string concatenation.
5. **Statement**
   * Used for SELECT queries when no parameters are needed.
6. **ResultSet**
   * Holds the data returned by SELECT queries and allows iteration over records.
7. **executeUpdate() vs executeQuery()**
   * executeUpdate() → For INSERT, UPDATE, DELETE, returns rows affected.
   * executeQuery() → For SELECT, returns ResultSet.
8. **Menu-driven Switch Case**
   * Allows user to choose operations dynamically without restarting the program.
9. **try-catch Block**
   * Handles SQL or Class errors gracefully.
10. **Connection.close() & Scanner.close()**
    * Prevents memory leaks by closing resources after use.

Flowchart:



Got it—you’re using a **plain Java project** without Maven or Gradle, so you won’t have a pom.xml file for dependency management.

Here’s what you need to do to use **MongoDB** in your project:

**Step 1: Download MongoDB Java Driver JAR**

1. Go to: [MongoDB Java Driver Download](https://repo1.maven.org/maven2/org/mongodb/mongodb-driver-sync/4.11.1/)
2. Download the following JAR files:
   * mongodb-driver-sync-4.11.1.jar
   * bson-4.11.1.jar
   * mongodb-driver-core-4.11.1.jar
3. Add them to your **Java Project → Build Path → Add External JARs**.

**Step 2: Java Code for MongoDB CRUD**

Here is the **complete code** for MongoDB CRUD in a simple Java project without Maven:

import com.mongodb.client.\*;

import org.bson.Document;

import java.util.Scanner;

import static com.mongodb.client.model.Filters.eq;

public class MongoDBCRUD {

public static void main(String[] args) {

// 1. Connect to MongoDB server

MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");

// 2. Access Database & Collection

MongoDatabase database = mongoClient.getDatabase("university");

MongoCollection<Document> collection = database.getCollection("engineeringstudents");

Scanner sc = new Scanner(System.in);

while (true) {

System.out.println("1. Add Student 2. View Students 3. Update Student 4. Delete Student 5. Exit");

int choice = sc.nextInt();

switch (choice) {

case 1: addStudent(collection, sc); break;

case 2: viewStudents(collection); break;

case 3: updateStudent(collection, sc); break;

case 4: deleteStudent(collection, sc); break;

case 5:

mongoClient.close();

sc.close();

return;

default: System.out.println("Invalid Choice!");

}

}

}

// CREATE

public static void addStudent(MongoCollection<Document> collection, Scanner sc) {

System.out.print("Enter ID: "); int id = sc.nextInt(); sc.nextLine();

System.out.print("Enter Department: "); String dept = sc.nextLine();

System.out.print("Enter First Name: "); String fname = sc.nextLine();

System.out.print("Enter Last Name: "); String lname = sc.nextLine();

System.out.print("Enter Passed Year: "); int year = sc.nextInt();

System.out.print("Enter University Rank: "); int rank = sc.nextInt();

Document doc = new Document("student\_id", id)

.append("department", dept)

.append("first\_name", fname)

.append("last\_name", lname)

.append("passed\_year", year)

.append("university\_rank", rank);

collection.insertOne(doc);

System.out.println("Student Added Successfully!");

}

// READ

public static void viewStudents(MongoCollection<Document> collection) {

for (Document doc : collection.find()) {

System.out.println(doc.toJson());

}

}

// UPDATE

public static void updateStudent(MongoCollection<Document> collection, Scanner sc) {

System.out.print("Enter Student ID to Update: "); int id = sc.nextInt();

System.out.print("Enter New University Rank: "); int rank = sc.nextInt();

var result = collection.updateOne(eq("student\_id", id),

new Document("$set", new Document("university\_rank", rank)));

if (result.getModifiedCount() > 0) System.out.println("Student Updated Successfully!");

else System.out.println("Student ID Not Found!");

}

// DELETE

public static void deleteStudent(MongoCollection<Document> collection, Scanner sc) {

System.out.print("Enter Student ID to Delete: "); int id = sc.nextInt();

var result = collection.deleteOne(eq("student\_id", id));

if (result.getDeletedCount() > 0) System.out.println("Student Deleted Successfully!");

else System.out.println("Student ID Not Found!");

}

}

**Step 3: Run the Program**

* Make sure MongoDB is running:
* mongod
* Create the university database and engineeringstudents collection automatically by inserting the first record.