

UNIT 2

Database Programming

(7 Hours) — Advanced Java Programming (CACS354)

Important Notice — Please Read

These notes were prepared from the syllabus and general knowledge of Java. **They may contain errors** (typos, outdated API details, or mistakes in code and explanations), so please take care. Verify important points and test all code yourself, and cross-check with your textbook (Core Java Vol. II / Java: The Complete Reference), the official Java documentation, and your teacher's notes.

2.1 The Design of JDBC

JDBC (Java Database Connectivity) is a Java API (packages `java.sql` and `javax.sql`) that lets Java programs connect to relational databases, send SQL statements and process the results. The application talks only to the JDBC API; a **JDBC driver** supplied by the database vendor translates the calls into the database's own protocol. This makes the application mostly independent of the database used.

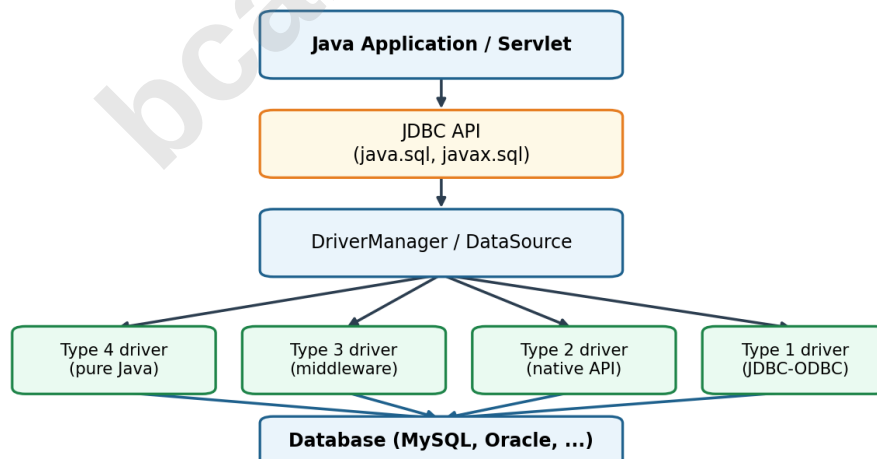


Figure 2.1 — JDBC architecture and driver types

2.1.1 JDBC Driver Types

Type	Name	How it works	Pros / Cons
Type 1	JDBC–ODBC bridge	Translates JDBC calls to ODBC calls; needs an ODBC driver on client	Easy but slow, platform dependent; removed in Java 8

Type	Name	How it works	Pros / Cons
Type 2	Native-API (partly Java)	Converts JDBC calls into native client library calls (e.g. Oracle OCI)	Faster than Type 1; needs native library installed on client
Type 3	Network-protocol (middleware)	JDBC calls go to a middleware server, which talks to the database	Flexible, no client install; extra server needed
Type 4	Thin / pure Java	Converts JDBC calls directly into the database's network protocol	Fastest and most portable; most widely used today (MySQL Connector/J, etc.)

2.1.2 Typical Uses of JDBC

- **Two-tier** (client/server): a Java application (e.g. Swing) connects directly to the database.
- **Three-tier**: client (browser/GUI) → middle tier (servlet, JSP, application server using JDBC) → database. This is the common web architecture.
- Enterprise and web applications such as banking, e-commerce, library and student-management systems.

2.2 The Structured Query Language (SQL)

SQL is the standard language for relational databases. JDBC simply passes SQL strings to the database. Main categories:

Category	Commands	Purpose
DDL	CREATE, ALTER, DROP	Define / change structure
DML	INSERT, UPDATE, DELETE	Modify data
DQL	SELECT	Query data
DCL / TCL	GRANT, REVOKE, COMMIT, ROLLBACK	Permissions and transactions

```
CREATE TABLE student (
    id      INT PRIMARY KEY,
    name    VARCHAR(50) NOT NULL,
    course  VARCHAR(20),
    marks   DOUBLE
);
INSERT INTO student VALUES (1, 'Ram', 'BCA', 78.5);
UPDATE student SET marks = 82 WHERE id = 1;
DELETE FROM student WHERE id = 1;
SELECT name, marks FROM student WHERE marks > 60 ORDER BY marks DESC;
```

2.3 JDBC Configuration

Before writing code you must set up the following:

- Install a database (MySQL, PostgreSQL, Oracle, etc.) and create a database and tables.
- Download the vendor's **JDBC driver JAR** (for MySQL: Connector/J) and add it to the project's classpath (or the `WEB-INF/lib` folder for web apps).
- Know the **driver class name** and the **JDBC URL**. Since JDBC 4.0 (Java 6) drivers on the classpath are auto-loaded, so `Class.forName(...)` is usually optional.

Database	Driver class	JDBC URL format
MySQL	<code>com.mysql.cj.jdbc.Driver</code>	<code>jdbc:mysql://localhost:3306/dbname</code>
PostgreSQL	<code>org.postgresql.Driver</code>	<code>jdbc:postgresql://localhost:5432/dbname</code>
Oracle	<code>oracle.jdbc.OracleDriver</code>	<code>jdbc:oracle:thin:@localhost:1521:xe</code>
SQLite	<code>org.sqlite.JDBC</code>	<code>jdbc:sqlite:test.db</code>

Note: Class names and URLs differ between driver versions (for example older MySQL drivers used `com.mysql.jdbc.Driver`). Check your driver's documentation.

2.4 Working with JDBC Statements

2.4.1 Steps to Connect and Query

1. Load / register the driver (optional in modern JDBC).
2. Establish a connection: `DriverManager.getConnection(url, user, password)`.
3. Create a statement: `Statement`, `PreparedStatement` or `CallableStatement`.
4. Execute the SQL: `executeQuery()` (SELECT) or `executeUpdate()` (INSERT/UPDATE/DELETE/DDDL).
5. Process the `ResultSet` (for queries).
6. Close `ResultSet`, `Statement` and `Connection` (best done using try-with-resources).

```
import java.sql.*;

public class FirstJdbc {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/college";
        try (Connection con = DriverManager.getConnection(url, "root", "password");
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("SELECT id, name, marks FROM student")) {

            while (rs.next()) {
                int id = rs.getInt("id");           // by column name
            }
        }
    }
}
```

```

        String name = rs.getString(2);           // by column index (1-based)
        double marks = rs.getDouble("marks");
        System.out.println(id + " " + name + " " + marks);
    }
} catch (SQLException e) {
    e.printStackTrace();
}
}
}

```

2.4.2 Types of Statements

Interface	Use	Notes
Statement	Simple static SQL	SQL is compiled each time; vulnerable to SQL injection if strings are concatenated
PreparedStatement	Precompiled SQL with ? parameters	Faster for repeated use; prevents SQL injection; preferred
CallableStatement	Calls stored procedures	Uses {call procName(?,?)} syntax

```

String sql = "INSERT INTO student(id, name, course, marks) VALUES (?, ?, ?, ?)";
try (Connection con = DriverManager.getConnection(url, user, pwd);
    PreparedStatement ps = con.prepareStatement(sql)) {
    ps.setInt(1, 5);
    ps.setString(2, "Sita");
    ps.setString(3, "BCA");
    ps.setDouble(4, 88.0);
    int rows = ps.executeUpdate();           // number of rows affected
    System.out.println(rows + " row(s) inserted");
}

// Stored procedure
CallableStatement cs = con.prepareCall("{call getMarks(?, ?)}");
cs.setInt(1, 5);                           // IN parameter
cs.registerOutParameter(2, Types.DOUBLE);  // OUT parameter
cs.execute();
double m = cs.getDouble(2);

```

2.5 Query Execution

Method	Returns	Used for
executeQuery(sql)	ResultSet	SELECT statements
executeUpdate(sql)	int (row count)	INSERT, UPDATE, DELETE, DDL
execute(sql)	boolean (true if ResultSet)	Unknown / multiple results; use getResultSet() or getUpdateCount()
executeBatch()	int[]	Runs many statements added by addBatch()

Transactions

By default every statement is committed automatically (*auto-commit*). For a group of statements that must succeed together, turn it off and commit or roll back manually.

```
con.setAutoCommit(false);
try {
    ps1.executeUpdate();           // debit account A
    ps2.executeUpdate();           // credit account B
    con.commit();                   // both succeeded
} catch (SQLException e) {
    con.rollback();                 // undo both
}
```

Handling Results — ResultSetMetaData

```
ResultSetMetaData md = rs.getMetaData();
int cols = md.getColumnCount();
for (int i = 1; i <= cols; i++)
    System.out.print(md.getColumnName(i) + "\t");
```

Example 2.1 — Student CRUD

Write the SQL operations needed for a student table: add, update marks, delete and list students.

Solution

Use PreparedStatement for each operation with ? placeholders, executeUpdate() for changes and executeQuery() for the list.

```
public class StudentDAO {
    private final String url = "jdbc:mysql://localhost:3306/college";
    private Connection get() throws SQLException {
        return DriverManager.getConnection(url, "root", "password");
    }
    public void add(int id, String n, double m) throws SQLException {
        try (Connection c = get();
             PreparedStatement p = c.prepareStatement("INSERT INTO student(id,name,marks) VALUES(?,?,?)")) {
            p.setInt(1, id); p.setString(2, n); p.setDouble(3, m); p.executeUpdate();
        }
    }
    public void updateMarks(int id, double m) throws SQLException {
        try (Connection c = get();
             PreparedStatement p = c.prepareStatement("UPDATE student SET marks=? WHERE id=?")) {
            p.setDouble(1, m); p.setInt(2, id); p.executeUpdate();
        }
    }
    public void delete(int id) throws SQLException {
        try (Connection c = get();
             PreparedStatement p = c.prepareStatement("DELETE FROM student WHERE id=?")) {
            p.setInt(1, id); p.executeUpdate();
        }
    }
}
```

2.6 Scrollable and Updatable Result Sets

A default **ResultSet** is **forward-only** and **read-only**. By passing flags to `createStatement` (or `prepareStatement`) you can make it scrollable and/or updatable.

Constant	Meaning
TYPE_FORWARD_ONLY	Cursor moves only forward (default)
TYPE_SCROLL_INSENSITIVE	Scrollable; does not reflect changes made by others after the query
TYPE_SCROLL_SENSITIVE	Scrollable; may reflect changes made by others
CONCUR_READ_ONLY	Result set cannot be used to update the database (default)
CONCUR_UPDATABLE	Result set can update the database

```
Statement st = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                   ResultSet.CONCUR_UPDATABLE);
ResultSet rs = st.executeQuery("SELECT id, name, marks FROM student");

rs.last(); // move to last row
System.out.println("Rows: " + rs.getRow());
rs.first(); rs.next(); rs.previous();
rs.absolute(3); // third row
rs.relative(-1); // one row back

// update current row
rs.updateDouble("marks", 91.0);
rs.updateRow(); // writes change to database

// insert a new row
rs.moveToInsertRow();
rs.updateInt("id", 10); rs.updateString("name", "Hari"); rs.updateDouble("marks", 70);
rs.insertRow();
rs.moveToCurrentRow();

rs.absolute(2); rs.deleteRow(); // delete row 2
```

Note: Whether sensitivity and updatability are really supported depends on the driver and database. A query must usually select from a single table and include the primary key to be updatable.

2.7 Row Sets

A **RowSet** (package `javax.sql.rowset`) is a JavaBeans-style component that extends **ResultSet**. It is scrollable and updatable by default, can manage its own connection, and supports listeners (events).

RowSet type	Description
JdbcRowSet	Connected RowSet; thin wrapper around ResultSet that makes it a JavaBean (keeps connection open)
CachedRowSet	Disconnected; caches data in memory, can be modified and synchronised back to the database later

RowSet type	Description
WebRowSet	Cached RowSet that can read/write its data as XML
FilteredRowSet	Cached RowSet that filters rows using a Predicate
JoinRowSet	Combines (joins) several RowSets

```
import javax.sql.rowset.*;

RowSetFactory factory = RowSetProvider.newFactory();
try (CachedRowSet crs = factory.createCachedRowSet()) {
    crs.setUrl("jdbc:mysql://localhost:3306/college");
    crs.setUsername("root");
    crs.setPassword("password");
    crs.setCommand("SELECT id, name, marks FROM student WHERE marks > ?");
    crs.setDouble(1, 50);
    crs.execute(); // connects, fetches, disconnects

    while (crs.next())
        System.out.println(crs.getString("name"));

    crs.absolute(1);
    crs.updateDouble("marks", 95);
    crs.updateRow();
    crs.acceptChanges(); // reconnect and write changes to DB
}
```

ResultSet	RowSet
Always created from a Statement	Can be configured and executed on its own
Connected to the database while in use	Cached versions are disconnected
Forward-only and read-only by default	Scrollable and updatable by default
Not a JavaBean	JavaBean with properties and listeners

2.8 Short Questions for Revision

- What is JDBC? Describe the four types of JDBC drivers.
- Differentiate Statement and PreparedStatement. Why is PreparedStatement safer?
- Write the steps to connect to a database with JDBC.
- What are scrollable and updatable result sets? Write a code fragment.
- What is a RowSet? Differentiate CachedRowSet and JdbcRowSet.
- Write a JDBC program to insert and display records from a table.

Key Takeaways — Unit 2

- ✓ JDBC is the standard Java API for relational databases; drivers (Type 1–4) translate calls for each DBMS, and Type 4 is most common.
- ✓ Steps: get Connection → create Statement → execute SQL → process ResultSet → close resources (try-with-resources).
- ✓ Prefer PreparedStatement (parameterised, precompiled, injection-safe); CallableStatement is for stored procedures.
- ✓ executeQuery() for SELECT; executeUpdate() for INSERT/UPDATE/DELETE; execute() for unknown results.
- ✓ Scrollable / updatable ResultSets are created with TYPE_SCROLL_* and CONCUR_UPDATABLE flags.
- ✓ RowSets are JavaBean-style, scrollable, updatable ResultSets; CachedRowSet works while disconnected.

— *End of Unit 2* —

Reminder: these notes may contain errors — verify against your textbook and run the code yourself.