

JAVA PROGRAMMING

CHAPTER 7 DATA BASE CONNECTIVITY : JDBC

Java as a Database Front End • Client/Server Methodology • Two-Tier and Three-Tier Design • The JDBC API • JDBC Drivers • JDBC-ODBC Bridge

Detailed Notes for Engineering Students

Includes Solved Examples, 15 MCQs with Answer Key and 10 Broad Questions

CONTENTS

7.1 Java as a Database Front End	3
7.1.1 Introduction	3
7.1.2 Why Java is Suitable as a Database Front End	3
7.1.3 Database Client/Server Methodology	4
7.1.4 Two-Tier Database Design	5
7.1.5 Three-Tier Database Design	6
7.2 The JDBC API	8
7.2.1 Introduction to JDBC	8
7.2.2 The API Components	8
7.2.3 Limitations of Using JDBC (Applications vs. Applets)	10
7.2.4 Security Considerations	11
7.2.5 A JDBC Database Example	12
7.2.6 JDBC Drivers	15
7.2.7 JDBC-ODBC Bridge	16
7.2.8 Current JDBC Drivers	16
Chapter Summary	18
MULTIPLE CHOICE QUESTIONS	19
Answer Key	20
BROAD QUESTIONS	21

CHAPTER 7

DATA BASE CONNECTIVITY : JDBC

Learning Objectives of this Chapter

After studying this chapter you will be able to: (1) explain why Java is used as a database front end; (2) describe client/server, two-tier and three-tier database architectures; (3) list and explain the components of the JDBC API; (4) discuss limitations and security issues of JDBC in applications and applets; (5) write a complete JDBC program using the standard steps; (6) classify JDBC drivers into four types and explain the JDBC-ODBC bridge.

7.1 Java as a Database Front End

7.1.1 Introduction

Almost every real-world application - a college admission system, a railway reservation portal, a bank, an online shop - needs to **store**, **retrieve** and **update** large amounts of data. This data is kept in a **database**, which is an organised collection of related data managed by software called a **Database Management System (DBMS)**. Popular relational DBMS (RDBMS) products include MySQL, Oracle, PostgreSQL, Microsoft SQL Server, IBM DB2 and SQLite. All of them are accessed using a common language called **SQL (Structured Query Language)**.

The database by itself has no user-friendly screen. Users need a program that accepts input, sends requests to the database and displays the results in a readable form. This program is called the **front end** (client), while the database that stores the data is called the **back end** (server). **Java** is widely used to build such front ends.

Term	Meaning
Front end	The part of the system the user sees and interacts with (forms, buttons, tables, web pages). Written in Java in our case.
Back end	The database server where data is physically stored, indexed and protected (MySQL, Oracle, etc.).
DBMS	Software that creates, manages and controls access to a database.
SQL	Standard language used to create tables and to insert, select, update and delete data.
Query	A SQL statement sent to the database to get or change data.
Result Set	The table of rows returned by a SELECT query.

7.1.2 Why Java is Suitable as a Database Front End

Java is not just a language for desktop programs; it provides a complete set of classes to talk to databases. The main reasons Java is preferred for database applications are given below.

- **Platform independence:** The same compiled Java program (bytecode) runs on Windows, Linux or macOS - 'write once, run anywhere'.
- **Standard database API:** JDBC gives one uniform set of classes and interfaces for all databases, so changing from MySQL to Oracle needs only small changes (driver and URL).
- **Object-oriented:** Table rows can be represented as objects (e.g., a *Student* class for the STUDENT table), making programs clean and reusable.
- **Robust and secure:** Strong type checking, automatic memory management, exception handling (*SQLException*) and the security manager.
- **Rich GUI and web support:** AWT, Swing, JavaFX, Applets, Servlets and JSP can all be used to design the user interface.
- **Networking and multithreading:** Built-in support helps in building multi-user, networked database applications.
- **Large community and free tools:** Many free drivers, IDEs and frameworks (JDBC, Hibernate, Spring) are available.

7.1.3 Database Client/Server Methodology

In the **client/server model**, the work of an application is divided between two cooperating programs that normally run on different computers connected by a network.

- The **client** is the program that *requests* a service. For us, it is the Java program (with or without a GUI).
- The **server** is the program that *provides* the service. For us, it is the database server (DBMS).
- The client sends a **request** (a SQL statement). The server processes it and sends back a **response** (rows of data or an update count).
- One server can serve many clients at the same time.

Responsibility	Client (Java Program)	Server (DBMS)
Main role	Interacts with the user, builds SQL requests	Stores and manages data, executes SQL
Input / Output	Takes user input, shows results	Receives queries, returns result sets
Processing	Validation, formatting, display logic	Searching, sorting, joining, indexing
Security	Login screen, input checks	User accounts, privileges, data protection
Concurrency	Handles one user	Handles many users together (transactions, locks)
Location	User's machine (desktop/browser)	Central machine (server room/cloud)

Working steps of a database client/server interaction:

- 1 The client program connects to the database server by supplying its address, user name and password.
- 2 The client sends a SQL request to the server.

- 3 The server parses, optimises and executes the SQL statement.
- 4 The server sends the result (rows or a count of affected rows) back to the client.
- 5 The client displays the result and finally closes the connection.

Advantages of client/server methodology:

- Data is centralised, so it is easy to back up, secure and keep consistent.
- Many users can share the same data simultaneously.
- Processing load is shared between client and server, giving better performance.
- Client and server can be upgraded or replaced independently.

7.1.4 Two-Tier Database Design

In a **two-tier architecture** the Java application (client) communicates **directly** with the database server. There is no intermediate program between them. The client contains both the user-interface code and the business logic (rules such as 'marks cannot be above 100'), and it sends SQL to the database through JDBC.



Fig. 7.1 Two-Tier Database Architecture

Features:

- Tier 1 (Client): GUI + application logic + JDBC code.
- Tier 2 (Server): DBMS holding the data.
- Communication is done through a JDBC driver over the network (or on the same machine).
- Typical examples: a college library desktop software, a small shop billing program, a lab project using Java Swing and MySQL.

Advantages	Disadvantages
Simple to design, develop and test.	Poor scalability - performance falls when many clients connect at the same time.
Fast response because there is no middle layer.	Business logic is placed in every client, so a change requires updating all clients (hard maintenance).
Low development cost; suitable for small projects.	Database login details may be stored on every client machine, which is a security risk.
Easy to understand for beginners.	Each client keeps its own database connection, so the server gets overloaded.

7.1.5 Three-Tier Database Design

A **three-tier architecture** inserts a **middle (application) tier** between the client and the database. The client never talks to the database directly; it talks only to the application server, and the application server talks to the database using JDBC.



Fig. 7.2 Three-Tier Database Architecture

Tier	Also called	Function	Example Technology
Tier 1	Presentation / Client tier	Displays the user interface and takes input from the user.	Web browser, Swing/JavaFX GUI, Applet, mobile app
Tier 2	Application / Business / Middle tier	Contains business rules, validation, security checks; builds and executes SQL using JDBC.	Servlets, JSP, Java application server, Spring
Tier 3	Data tier	Stores, retrieves and protects the data.	MySQL, Oracle, PostgreSQL, SQL Server

Advantages of the three-tier design:

- **Scalability:** Many clients share a pool of database connections held by the middle tier.
- **Security:** The database is hidden behind the application server; clients never see the database user name or password.
- **Easy maintenance:** Business logic is kept in one place; changes do not require reinstalling the client.
- **Reusability and flexibility:** One middle tier can serve web, desktop and mobile clients; the database can be replaced without changing the client.
- **Applets can reach any database:** An applet can connect only to its own host, so the middle tier on that host connects onward to the database.

Disadvantages: more complex design, more software to install and manage, and slightly slower because of the extra layer.

Comparison: Two-Tier vs Three-Tier

Basis	Two-Tier	Three-Tier
Number of layers	Client and Database	Client, Application server and Database
Client-to-database link	Direct	Indirect, through the middle tier

Basis	Two-Tier	Three-Tier
Business logic location	Inside the client	Inside the application server
Scalability	Low	High
Security	Lower	Higher
Maintenance	Difficult (update every client)	Easy (update only server)
Cost and complexity	Low	Higher
Best suited for	Small desktop projects, labs	Large web and enterprise applications

Remember: In **n-tier** architecture more layers (for example a separate web server, application server, and database server) can be added. Three-tier is the most common form used in industry.

7.2 The JDBC API

7.2.1 Introduction to JDBC

JDBC (Java Database Connectivity) is a standard Java API that allows a Java program to connect to a relational database, send SQL statements and process the results. It was introduced by Sun Microsystems in JDK 1.1 (1997) and is now maintained by Oracle. JDBC is not a database; it is a **set of interfaces and classes** in the packages **java.sql** (core API) and **javax.sql** (extended API: DataSource, connection pooling, RowSet).

JDBC works in a similar way to ODBC (Open Database Connectivity) of Microsoft, but it is written in Java and is therefore platform independent. JDBC defines the *interfaces*; each database vendor supplies a **JDBC driver** that *implements* those interfaces for its own database. Your program is written against the interfaces only, so it stays almost unchanged when the database is changed.

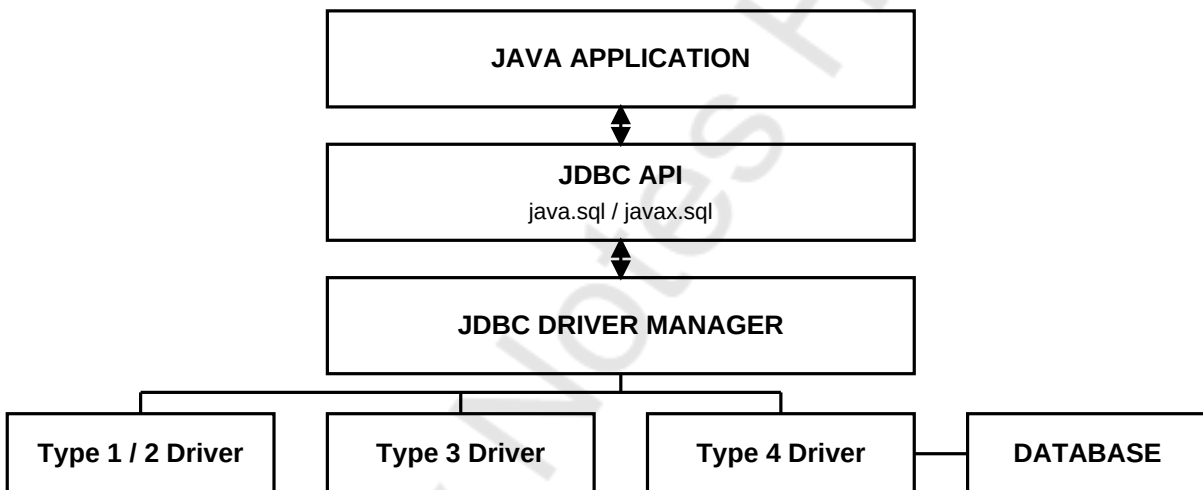


Fig. 7.3 JDBC Architecture

Main tasks performed by JDBC:

- Establish a connection with a database.
- Send SQL statements (query or update).
- Retrieve and process the results.
- Handle errors and close the connection.

7.2.2 The API Components

The JDBC API is a collection of interfaces and classes. The important components and their roles are summarised below.

Component	Package type	Purpose
DriverManager	Class	Manages the list of registered drivers and creates connections through <code>getConnection(url, user, password)</code> .

Component	Package type	Purpose
Driver	Interface	Implemented by each vendor's driver; knows how to talk to one particular database.
Connection	Interface	Represents one session with the database. Used to create statements and to control transactions (<i>commit()</i> , <i>rollback()</i> , <i>setAutoCommit()</i> , <i>close()</i>).
Statement	Interface	Executes a simple, static SQL string. Methods: <i>executeQuery()</i> , <i>executeUpdate()</i> , <i>execute()</i> .
PreparedStatement	Interface	Precompiled SQL with ? placeholders for parameters; faster for repeated use and safe against SQL injection.
CallableStatement	Interface	Used to call stored procedures and functions in the database.
ResultSet	Interface	Table of data returned by a SELECT. Moved row by row with <i>next()</i> ; values read with <i>getInt()</i> , <i>getString()</i> , <i>getDouble()</i> , etc.
ResultSetMetaData	Interface	Information about the columns of a ResultSet (count, names, types).
DatabaseMetaData	Interface	Information about the database itself (product name, version, tables, supported features).
SQLException	Class	Exception thrown for any database access error; has <i>getMessage()</i> , <i>getSQLState()</i> , <i>getErrorCode()</i> .
Types / Date / Time / Timestamp	Classes	Map SQL data types (VARCHAR, INTEGER, DATE ...) to Java types.
DataSource (javax.sql)	Interface	Preferred way to obtain connections in server applications; supports connection pooling.

Important Statement methods:

Method	Used for	Return value
<i>executeQuery(sql)</i>	SELECT statements	ResultSet object
<i>executeUpdate(sql)</i>	INSERT, UPDATE, DELETE, CREATE, DROP	int - number of rows affected (0 for DDL)
<i>execute(sql)</i>	Any SQL when the type is not known	boolean - true if the result is a ResultSet

Difference between Statement, PreparedStatement and CallableStatement:

Feature	Statement	PreparedStatement	CallableStatement
Use	Simple static SQL	Parameterised SQL executed many times	Stored procedures
Precompiled	No	Yes	Yes (stored in DB)
Parameters	Not supported	IN parameters using ?	IN, OUT and INOUT
SQL injection safety	Poor	Good	Good

Feature	Statement	PreparedStatement	CallableStatement
Speed for repeated use	Slow	Fast	Fast

Steps to access a database using JDBC (the seven standard steps):

- 1 Import the *java.sql* package.
- 2 Load and register the JDBC driver (automatic in JDBC 4.0 and later if the driver JAR is on the classpath).
- 3 Establish a connection using *DriverManager.getConnection()*.
- 4 Create a Statement (or PreparedStatement) object.
- 5 Execute the SQL query or update.
- 6 Process the results (loop through the ResultSet).
- 7 Close the ResultSet, Statement and Connection.

7.2.3 Limitations of Using JDBC (Applications vs. Applets)

JDBC can be used from two kinds of Java programs: **stand-alone applications** and **applets** (small programs downloaded from a web server and run inside a browser). The two differ greatly because of the **applet security sandbox**.

Point	Java Application	Java Applet
Where it runs	Directly on user's computer using JVM	Inside a web browser/applet viewer
Database connection	Can connect to any database host on the network	Can connect only to the host from which the applet was downloaded (unless signed)
Local file access	Allowed	Not allowed by default
Driver type usable	Any of the four types	Mainly Type 3 or Type 4 (pure Java). Types 1 and 2 need native code on the client, which is normally forbidden
Driver installation	Driver JAR must be in the classpath on that machine	Driver must be downloaded with the applet or already on the client
Security manager	None by default	Restrictive security manager is always active
Typical design	Two-tier	Three-tier (applet talks to a server which talks to the database)

Other general limitations of JDBC:

- JDBC works with **relational (SQL) databases** only; it is not designed for all NoSQL stores.
- **Vendor differences:** SQL dialects, data types and some features differ among databases, so a program is not always 100% portable.

- **Performance overhead:** Each call passes through driver layers; a Type 1 (bridge) driver is the slowest because of the double translation.
- **Low-level and verbose:** Opening/closing connections, mapping rows to objects and exception handling must be coded manually (frameworks like Hibernate/JPA reduce this).
- **Network dependency:** A broken connection raises *SQLException*; the program must handle it.
- **Resource leaks:** Forgetting to close Connection, Statement or ResultSet wastes database resources.
- **Applets are obsolete:** Modern browsers have removed applet support, so today JDBC is mainly used from applications, servlets and server-side code.

7.2.4 Security Considerations

Because a database often holds sensitive information (marks, salaries, bank balances), JDBC programs must be written with security in mind. The important points are as follows.

Security Issue	Explanation	Precaution
Applet sandbox	Untrusted applets are not allowed to connect to arbitrary hosts or read local files.	Use a three-tier design or digitally signed applets when more access is needed.
Security Manager	The Java SecurityManager checks permissions for network and file operations.	Grant only the permissions that are really needed (policy file).
SQL injection	Attacker types SQL code in a text box, e.g. ' OR '1'='1, to change the meaning of a query built by string joining.	Always use PreparedStatement with ? placeholders; validate all input.
Hard-coded passwords	User name and password written in the source or class file can be read by others.	Keep credentials in encrypted configuration files or environment variables; use a three-tier design.
Network eavesdropping	Data sent between client and database can be intercepted.	Use SSL/TLS encrypted connections.
Excess privileges	Using the 'root' or 'admin' account for every program is dangerous.	Create database users with minimum privileges (principle of least privilege).
Error messages	Printing the full <i>SQLException</i> stack trace to users reveals table and column names.	Log details privately; show a simple message to the user.
Unclosed resources	Open connections can be misused or exhaust the server.	Close resources in <i>finally</i> or use try-with-resources.

Example of SQL injection and its prevention:

```
// UNSAFE - query is built by joining strings
String sql = "SELECT * FROM users WHERE name='" + name + "' AND pass='" + pass + "'";

// SAFE - parameters are sent separately
String sql = "SELECT * FROM users WHERE name=? AND pass=?";
PreparedStatement ps = con.prepareStatement(sql);
ps.setString(1, name);
ps.setString(2, pass);
ResultSet rs = ps.executeQuery();
```

7.2.5 A JDBC Database Example

Let us create a small database named **college** having a table **student**. We will insert a record and then display all records using JDBC. The database used is MySQL, but the same program works with any database after changing the driver and URL.

Step A: Create the table (SQL)

```
CREATE DATABASE college;
USE college;
CREATE TABLE student (
    roll    INT PRIMARY KEY,
    name    VARCHAR(40),
    branch  VARCHAR(20),
    marks   INT
);
INSERT INTO student VALUES (1, 'Amit', 'CSE', 78);
INSERT INTO student VALUES (2, 'Riya', 'ECE', 85);
```

Step B: Java program

```

import java.sql.*;

public class StudentDemo {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/college";
        String user = "root";
        String pass = "password";

        try {
            // 1. Load driver (optional in JDBC 4.0+)
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Connect
            Connection con = DriverManager.getConnection(url, user, pass);

            // 3. Insert a record using PreparedStatement
            String ins = "INSERT INTO student VALUES (?, ?, ?, ?)";
            PreparedStatement ps = con.prepareStatement(ins);
            ps.setInt(1, 3);
            ps.setString(2, "Sohan");
            ps.setString(3, "ME");
            ps.setInt(4, 69);
            int n = ps.executeUpdate();
            System.out.println(n + " row inserted");

            // 4. Retrieve records
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("SELECT * FROM student");
            while (rs.next()) {
                System.out.println(rs.getInt("roll") + " "
                                   + rs.getString("name") + " "
                                   + rs.getString("branch") + " "
                                   + rs.getInt("marks"));
            }

            // 5. Close resources
            rs.close(); st.close(); ps.close(); con.close();

        } catch (ClassNotFoundException e) {
            System.out.println("Driver not found: " + e);
        } catch (SQLException e) {
            System.out.println("Database error: " + e.getMessage());
        }
    }
}

```

Output:

```

1 row inserted
1 Amit CSE 78
2 Riya ECE 85
3 Sohan ME 69

```

Explanation of the program:

Line / Statement	Explanation
import java.sql.*;	Imports all JDBC classes and interfaces.
Class.forName(...)	Loads the driver class into memory so it registers itself with DriverManager. Not required for JDBC 4.0+ drivers found on the classpath.
jdbc:mysql://localhost:3306/college	JDBC URL in the form <i>jdbc:subprotocol://host:port/database</i> .
DriverManager method getConnection()	Opens a session with the database and returns a Connection object.
prepareStatement(...)	Creates a precompiled statement; ? marks are replaced using setInt() and setString() (index starts at 1).
executeUpdate()	Runs INSERT/UPDATE/DELETE and returns the number of rows affected.
executeQuery()	Runs a SELECT and returns a ResultSet.
rs.next()	Moves the cursor to the next row; returns false when no rows are left. The cursor initially sits before the first row.
rs.getInt("roll")	Reads a column value by name (or by position, e.g. getInt(1)).
close()	Releases database resources.

Modern style - try-with-resources (Java 7 and above):

```
try (Connection con = DriverManager.getConnection(url, user, pass);
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT name, marks FROM student")) {

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

} catch (SQLException e) {
    e.printStackTrace();
}
// Connection, Statement and ResultSet are closed automatically.
```

JDBC URL formats:

Database	Driver class	URL format
MySQL	com.mysql.cj.jdbc.Driver	<i>jdbc:mysql://host:3306/dbname</i>
Oracle	oracle.jdbc.OracleDriver	<i>jdbc:oracle:thin:@host:1521:SID</i>
PostgreSQL	org.postgresql.Driver	<i>jdbc:postgresql://host:5432/dbname</i>
SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver	<i>jdbc:sqlserver://host:1433;databaseName=db</i>
SQLite	org.sqlite.JDBC	<i>jdbc:sqlite:path/to/file.db</i>
ODBC bridge (old)	sun.jdbc.odbc.JdbcOdbcDriver	<i>jdbc:odbc:DataSourceName</i>

7.2.6 JDBC Drivers

A **JDBC driver** is a software component (usually a JAR file) that translates JDBC calls made by the Java program into the protocol understood by a specific database. Without a driver a Java program cannot communicate with a database. Sun classified JDBC drivers into **four types**.

Type 1 - JDBC-ODBC Bridge Driver

- Converts JDBC calls into ODBC calls; the ODBC driver then talks to the database.
- Needs ODBC driver and native code installed on every client machine.
- Slow because of double translation; not suitable for applets or production use.

Type 2 - Native-API (Partly Java) Driver

- Converts JDBC calls into the database's native client library calls (for example Oracle OCI).
- Faster than Type 1, but native client software must still be installed on every client.
- Not portable; cannot be used in applets.

Type 3 - Network-Protocol (Middleware) Driver

- Pure Java client that sends JDBC calls over the network to a middleware server in a database-independent protocol; the middleware converts them to the database protocol.
- No client-side installation of database software; one driver can reach many databases; good for applets and three-tier systems.
- Needs an extra middleware server, which adds cost and maintenance.

Type 4 - Native-Protocol (Thin / Pure Java) Driver

- Converts JDBC calls directly into the network protocol of the specific database.
- 100% Java, no native code, platform independent, easily downloaded; best performance for most uses.
- A different driver is needed for each database (MySQL, Oracle, etc.). This is the most widely used type today.

Comparison of the four driver types:

Feature	Type 1	Type 2	Type 3	Type 4
Name	JDBC-ODBC Bridge	Native-API	Network Protocol	Native Protocol (Thin)
Written in	Java + native ODBC	Java + native code	Pure Java	Pure Java
Client software needed	ODBC driver	DB client library	None	None
Middleware server	No	No	Yes	No
Speed	Slowest	Fast	Medium	Fastest

Feature	Type 1	Type 2	Type 3	Type 4
Portability	Low	Low	High	High
Usable in applets	No	No	Yes	Yes
Typical use	Testing / old systems	Where vendor library exists	Enterprise / multi-database	Almost all applications

7.2.7 JDBC-ODBC Bridge

ODBC (Open Database Connectivity) is an older, language-independent C-based API from Microsoft for accessing databases. Almost every database, and even files such as MS Access, Excel and text files, has an ODBC driver. The **JDBC-ODBC Bridge** was a Type 1 driver supplied with the JDK (*sun.jdbc.odbc.JdbcOdbcDriver*) so that Java programs could reach any ODBC data source.

Working: Java program → JDBC API → JDBC-ODBC bridge → ODBC driver manager → ODBC driver → Database.

Steps to use the bridge (older Java versions):

- 1 Create a Data Source Name (DSN) in Windows: Control Panel → Administrative Tools → ODBC Data Sources, e.g. *mydsn*.
- 2 Load the driver: *Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");*
- 3 Connect: *DriverManager.getConnection("jdbc:odbc:mydsn", user, pass);*
- 4 Create statements and process results as usual.

Advantages	Disadvantages
Gave access to almost any database that has an ODBC driver, including MS Access and Excel.	Slow: JDBC call is converted to ODBC call and then to database call.
Useful for quick learning and testing when no other driver was available.	ODBC driver and DSN must be installed on every client; works mainly on Windows.
Easy to set up for small projects.	Not portable, not pure Java, not suitable for applets because of native code.
	Does not support all JDBC features; marked as experimental and not for production by Sun.

Important: The JDBC-ODBC Bridge was **removed from the JDK in Java 8**. For new programs, use the vendor's Type 4 driver (for example, UCanAccess for MS Access files, MySQL Connector/J for MySQL).

7.2.8 Current JDBC Drivers

Today the JDBC market is dominated by **Type 4 (pure Java) drivers** supplied free by database vendors or open-source communities. Since **JDBC 4.0** (Java 6), drivers are discovered

automatically through the Java Service Provider mechanism: if the driver JAR is on the classpath, *Class.forName()* is no longer necessary. Drivers are added to a project by placing the JAR in the classpath or, in build tools like Maven/Gradle, by adding a dependency.

Database	Current Driver	Type	Remarks
MySQL / MariaDB	MySQL Connector/J (mysql-connector-j); MariaDB Connector/J	4	Class: com.mysql.cj.jdbc.Driver (older: com.mysql.jdbc.Driver)
Oracle	Oracle JDBC Thin (ojdbc8, ojdbc11)	4 (Thin) / 2 (OCI)	Class: oracle.jdbc.OracleDriver; OCI variant needs Oracle client
PostgreSQL	PostgreSQL JDBC Driver (pgjdbc)	4	Open source; class: org.postgresql.Driver
Microsoft SQL Server	Microsoft JDBC Driver for SQL Server (mssql-jdbc)	4	Class: com.microsoft.sqlserver.jdbc.SQLServerDriver
SQLite	SQLite JDBC (xerial sqlite-jdbc)	4	Embedded, file-based database; no server needed
IBM DB2	IBM Data Server Driver for JDBC (JCC)	4 / 2	Both type versions available
MS Access	UCanAccess	4	Replacement for the removed JDBC-ODBC bridge

JDBC versions in brief:

- **JDBC 1.0 (JDK 1.1):** basic connection, statements and result sets.
- **JDBC 2.0 (JDK 1.2):** scrollable and updatable ResultSets, batch updates, javax.sql package, connection pooling, DataSource.
- **JDBC 3.0 (JDK 1.4):** savepoints, auto-generated keys, ParameterMetaData.
- **JDBC 4.0 (Java 6):** automatic driver loading, SQLXML, improved exception handling, annotations.
- **JDBC 4.1 to 4.3 (Java 7 to 9):** try-with-resources support, RowSetFactory, enhanced large update counts and sharding support.

Chapter Summary

Topic	Key Points to Remember
Front end / Back end	Java program is the front end (client); the database is the back end (server).
Two-tier	Client directly connected to database server; simple, but not scalable or very secure.
Three-tier	Client - Application server - Database; scalable, secure, easier to maintain.
JDBC	Java API (java.sql, javax.sql) for connecting to relational databases using SQL.
Main components	DriverManager, Connection, Statement, PreparedStatement, CallableStatement, ResultSet, SQLException.
Steps	Import, load driver, connect, create statement, execute, process results, close.
Applets vs applications	Applets are limited by the sandbox and can connect only to their origin host.
Security	Use PreparedStatement, least-privilege accounts, SSL, signed applets, no hard-coded passwords.
Driver types	Type 1 Bridge, Type 2 Native-API, Type 3 Network-Protocol, Type 4 Pure Java (Thin).
JDBC-ODBC Bridge	Type 1 driver; slow; removed from Java 8.
Current drivers	Vendor-supplied Type 4 JARs (Connector/J, ojdbc, pgjdbc, mssql-jdbc).

MULTIPLE CHOICE QUESTIONS

Choose the correct option for each question. The answer key is given at the end of this section.

Q1. What does JDBC stand for?

- (A) Java Data Binding Component (B) Java Database Connectivity (C) Java Debug Client (D) Joint Database Connector

Q2. Which package contains the core JDBC API classes and interfaces?

- (A) java.db (B) java.jdbc (C) javax.database (D) java.sql

Q3. In a two-tier database architecture, the Java client communicates directly with the:

- (A) Database server (B) Application server (C) Web server (D) Proxy server

Q4. In a three-tier architecture, the business logic is normally placed in the:

- (A) Client tier (B) Data tier (C) Middle (application) tier (D) Network layer

Q5. Which type of JDBC driver is known as the 'Thin' or pure Java driver?

- (A) Type 1 (B) Type 4 (C) Type 2 (D) Type 3

Q6. The JDBC-ODBC Bridge driver belongs to which driver type?

- (A) Type 1 (B) Type 2 (C) Type 3 (D) Type 4

Q7. Which interface is used to execute precompiled SQL statements with parameters?

- (A) Statement (B) CallableStatement (C) ResultSetMetaData (D) PreparedStatement

Q8. Which method is used to establish a connection with the database?

- (A) Connection.open() (B) Statement.connect() (C) DriverManager method getConnection() (D) Driver.start()

Q9. Which method of Statement is used to execute a SELECT query?

- (A) executeUpdate() (B) executeQuery() (C) executeSelect() (D) runQuery()

Q10. Which of the following best prevents SQL injection?

- (A) Using PreparedStatement (B) Using Statement with string joining (C) Using root account (D) Closing the connection late

Q11. By default, an untrusted applet can open a database connection only to:

- (A) Any host on the Internet (B) Any host on the local network (C) The host from which it was downloaded (D) No host at all

Q12. The JDBC-ODBC Bridge was removed from the JDK in:

- (A) Java 1.4 (B) Java 5 (C) Java 6 (D) Java 8

Q13. The method `ResultSet.next()` returns:

- (A) The next column value (B) true if another row is available, otherwise false (C) The number of rows (D) The previous row

Q14. Which interface is used to call stored procedures from JDBC?

- (A) `CallableStatement` (B) `PreparedStatement` (C) `Statement` (D) `DatabaseMetaData`

Q15. The driver class name for MySQL Connector/J 8.x is:

- (A) `sun.jdbc.odbc.JdbcOdbcDriver` (B) `oracle.jdbc.OracleDriver` (C) `com.mysql.cj.jdbc.Driver` (D) `org.postgresql.Driver`

Answer Key

Q.	Ans.	Q.	Ans.	Q.	Ans.
1	B	6	A	11	C
2	D	7	D	12	D
3	A	8	C	13	B
4	C	9	B	14	A
5	B	10	A	15	C

Answers with options: 1-B (Java Database Connectivity); 2-D (`java.sql`); 3-A (Database server); 4-C (Middle (application) tier); 5-B (Type 4); 6-A (Type 1); 7-D (`PreparedStatement`); 8-C (`DriverManager` method `getConnection()`); 9-B (`executeQuery()`); 10-A (Using `PreparedStatement`); 11-C (The host from which it was downloaded); 12-D (Java 8); 13-B (true if another row is available, otherwise false); 14-A (`CallableStatement`); 15-C (`com.mysql.cj.jdbc.Driver`).

BROAD QUESTIONS

Answer the following questions in detail (suggested marks: 5 to 10 each).

- 1 Explain the database client/server methodology. Discuss why Java is considered a suitable front end for database applications.
- 2 What is a two-tier database architecture? Draw its diagram and explain with advantages and disadvantages.
- 3 Explain the three-tier database architecture with a neat diagram. Compare it with the two-tier architecture.
- 4 What is JDBC? Describe the main components of the JDBC API with the role of each.
- 5 Write the steps required to connect a Java program to a database using JDBC. Write a complete program to insert and display records of a STUDENT table.
- 6 Differentiate between Statement, PreparedStatement and CallableStatement with suitable examples.
- 7 Discuss the limitations of using JDBC in applications and applets. Why can an applet connect only to its own host?
- 8 Explain the security considerations in JDBC programming. How can SQL injection be prevented?
- 9 Describe the four types of JDBC drivers with advantages, disadvantages and a comparison table.
- 10 Explain the JDBC-ODBC bridge with its working, advantages and disadvantages. List some current JDBC drivers used for popular databases.

- End of Chapter 7 -