Chapter 1 — Introduction to JDBC

Q: What is JDBC and why do we use it?

- JDBC (Java Database Connectivity) is a standard API that enables Java applications to connect with relational databases like MySQL, PostgreSQL, or Oracle.
- It allows developers to execute SQL queries directly from Java programs to insert, update, delete, or fetch data.
- It acts as a bridge between Java code and the database server, simplifying communication.
- In an e-commerce system, JDBC is used to fetch product lists, process orders, or verify user login credentials.

Q: How did JDBC evolve over time?

- JDBC was introduced in Java 1.1 (1997) by Sun Microsystems as the first database access API for Java.
- Over time, versions like JDBC 2.0 introduced batch updates and scrollable result sets.
- JDBC 3.0 added savepoints and auto-generated key retrieval, while JDBC 4.0 brought automatic resource management.
- The latest versions support modern data types like LocalDate and LocalDateTime, improving compatibility with Java 8+ features.

Q: Can you explain the JDBC architecture?

- The JDBC architecture has two main parts — the JDBC API and the JDBC Driver.
- The API provides interfaces such as Connection, Statement, and ResultSet to manage database operations.
- The Driver converts these method calls into database-specific commands.
- This separation ensures Java code remains independent of the database vendor.

Q: What are the key components of the JDBC API?

- DriverManager – manages the list of database drivers and establishes connections.

- Connection – represents an active link between the Java application and the database.
- Statement and PreparedStatement – used to execute SQL queries and updates.
- ResultSet – stores the data returned from SELECT queries.
- SQLException – handles database-related errors.

Q: How does JDBC make Java applications database-independent?

- JDBC defines a common set of interfaces that all database vendors implement via their drivers.
- The Java application interacts with the JDBC API, not the database directly.
- By changing the driver or connection URL, the same code can run on MySQL, Oracle, or PostgreSQL.
- This makes applications portable and easier to maintain across different environments.

Q: What are the different types of JDBC drivers?

- Type 1: JDBC-ODBC Bridge Driver – uses ODBC, now outdated and removed.
- Type 2: Native-API Driver – relies on native code libraries specific to each database.
- Type 3: Network Protocol Driver – uses a middleware server to communicate with databases.
- Type 4: Pure Java Driver – converts JDBC calls directly into database-specific protocol; it's the most widely used today.

Q: How do you load and register a JDBC driver?

- The driver must be loaded into memory before it can be used by DriverManager.
- This is typically done using the Class.forName() method.
- Example: Class.forName('com.mysql.cj.jdbc.Driver');
- Modern JDBC drivers auto-register themselves, but manual loading is still used for clarity in older systems.

Q: How does JDBC execute a SQL query behind the scenes?

- When a query is passed to executeQuery() or executeUpdate(), the JDBC driver sends it to the database.
- The driver translates the Java call into the database’s native protocol.
- The database executes the command and returns the result as a ResultSet.
- The Java program iterates through the ResultSet to process data such as product lists or customer orders.

Q: Why is it important to close JDBC resources?

- Each open connection consumes memory and database resources.
- If resources like Connection, Statement, or ResultSet are not closed, they can lead to memory leaks.
- In high-traffic applications like e-commerce, unclosed connections can exhaust the database connection pool.
- Using try-with-resources automatically closes these resources when the block ends.

Q: How do modern frameworks like Spring or Hibernate use JDBC internally?

- Spring JDBC and Hibernate both rely on JDBC underneath to interact with databases.
- Spring’s JdbcTemplate simplifies the code by handling connections and exceptions internally.
- Hibernate uses JDBC to execute SQL statements that are generated from Java entities.
- Even though these frameworks abstract the low-level details, JDBC remains the foundation for all data access operations.