Chapter 4 — Executing SQL Statements

Q: What are SQL commands and how are they categorized?

- SQL (Structured Query Language) is used to communicate with databases for creating, reading, updating, and deleting data.
- SQL commands are categorized into five main types: DDL (Data Definition Language), DML (Data Manipulation Language), DQL (Data Query Language), DCL (Data Control Language), and TCL (Transaction Control Language).
- DDL defines database structure (CREATE, ALTER, DROP), DML modifies data (INSERT, UPDATE, DELETE), and DQL retrieves data (SELECT).
- DCL manages permissions (GRANT, REVOKE), while TCL handles transaction control (COMMIT, ROLLBACK, SAVEPOINT).
- Each category serves a specific purpose to manage data and structure efficiently.

Q: What is the Statement interface in JDBC?

- The Statement interface is used to execute static SQL statements that do not require parameters.
- It is suitable for executing simple SQL operations like fetching or updating data in a table.
- Common methods include executeQuery(String sql), executeUpdate(String sql), and execute(String sql).
- Statements are ideal for queries that remain the same throughout execution.

Q: How do you execute a DDL statement using JDBC?

- DDL statements define or modify the structure of a database, such as creating or dropping tables.
- Use the executeUpdate() method of the Statement interface to execute DDL commands.
- Example: String sql = 'CREATE TABLE products (id INT PRIMARY KEY, name VARCHAR(50), price DOUBLE)'; statement.executeUpdate(sql);
- DDL statements usually return 0 since they affect schema, not rows, and are auto-committed by default.

Q: How do you execute a DML statement using JDBC?

- DML statements modify the data stored in the database (INSERT, UPDATE, DELETE).
- They are executed using the executeUpdate() method which returns the number of affected rows.
- Example: String insertSQL = 'INSERT INTO products (name, price) VALUES ('Laptop', 75000)'; int rows = statement.executeUpdate(insertSQL);
- Use transactions when executing multiple DML statements to ensure data integrity.

Q: How do you execute a SELECT query and process the ResultSet?

- Use executeQuery() to run SELECT statements that fetch data from tables.
- It returns a ResultSet object which can be iterated using a while loop to access each row.
- Example: ResultSet rs = statement.executeQuery('SELECT * FROM products'); while(rs.next()) { System.out.println(rs.getInt('id') + ' ' + rs.getString('name')); }
- Always close the ResultSet after processing to prevent memory leaks.

Q: What are some common SQL exceptions that occur during statement execution?

- SQLException: General exception for database access errors.
- SQLSyntaxErrorException: Triggered by incorrect SQL syntax.
- SQLIntegrityConstraintViolationException: Occurs when constraints like primary or foreign keys are violated.
- SQLTimeoutException: Thrown when query execution exceeds the timeout limit.
- Always handle exceptions with try-catch blocks and log errors for debugging.

Q: When should you use execute(), executeQuery(), and executeUpdate()?

- executeQuery(): Used for SELECT statements that return a ResultSet.
- executeUpdate(): Used for INSERT, UPDATE, DELETE, and DDL operations that modify data or schema.
- execute(): Used for SQL statements that may return multiple results or types (e.g., stored procedures).
- Choosing the right method improves performance and ensures correct results.

Q: How do you handle SQL exceptions properly while executing queries?

- Wrap JDBC operations inside try-catch blocks to handle runtime errors gracefully.
- Catch SQLException and log details like SQLState, error code, and message for debugging.
- Use logging frameworks like SLF4J or Log4j instead of System.out.println for production-grade applications.
- Example: catch(SQLException e) { logger.error('Error Code: {}, Message: {}', e.getErrorCode(), e.getMessage()); }

Q: Why is using PreparedStatement preferred over Statement for executing SQL?

- PreparedStatement is precompiled and provides better performance for repeated queries.
- It prevents SQL injection by separating SQL logic from user input.
- Allows the use of placeholders (?) to insert dynamic parameters safely.
- Example: PreparedStatement ps = con.prepareStatement('SELECT * FROM users WHERE email = ?'); ps.setString(1, 'test@example.com');
- PreparedStatements enhance both performance and security, especially in web applications.

Q: What are some best practices when executing SQL statements in JDBC?

- Always close resources like ResultSet, Statement, and Connection after use.
- Use try-with-resources to automatically manage resource cleanup.
- Prefer PreparedStatements over Statements for efficiency and security.
- Keep SQL logic simple and externalize complex queries in configuration files if possible.
- Log SQL execution times in production to detect slow-running queries and optimize performance.