**Java Industry Assignment**

Module 2– Java–RDBMS & Database Programming with JDBC

1. Introduction to JDBC

* What is JDBC (Java Database Connectivity)?
* JDBCstands for Java Database Connectivity.
* JDBCis a Java API to connect and execute the query with the database.
* It is a specification from Sun Microsystems that provides a standard abstraction (API or Protocol) for Java applications to communicate with various databases.
* It is used to write programs required to access databases.
* JDBC, along with the database driver, can access databases and spreadsheets.
* The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.
* Importance of JDBC in Java Programming
  + JDBC helps you to write Java applications that manage these three programming activities:
    1. Connect to a data source, like a database.
    2. Send queries and update statements to the database.
    3. Retrieve and process the results received from the database in answer to your query.
* JDBC Architecture: Driver Manager, Driver, Connection, Statement, and ResultSet
  + **Driver Manager:**This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
  + **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use Driver Manager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
  + **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
  + **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
  + **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

2. JDBC Driver Types

* Overview of JDBC Driver Types:
  + **Type1: JDBC-ODBC Bridge Driver:** The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver
* Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.
* **Type2: Native-API Driver:** The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.
* **Type3: Network Protocol Driver:** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.
* **Type4: Thin Driver:** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.
* Comparison and Usage of Each Driver Type:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Type 1 (JDBC-ODBC Bridge)** | **Type 2 (Native-API)** | **Type 3 (Network Protocol)** | **Type 4 (Thin Driver)** |
| **Performance** | slow (immediate layer) | Moderate | Moderate | High |
| **Portability** | Low (Platform Independent) | Low | High | High |
| **Ease of use** | Easy | Moderate | Complex | Easy |
| **Best use case** | Legacy Systems | Standalone Apps | Multi Tier Applications | Morden Applications |
| **Data Independence** | High | Low | High | Low |

3. Steps for Creating JDBC Connections

* Step-by-Step Process to Establish a JDBC Connection:
  1. **Import the JDBC Packages**: Include the java. SQL package to work with JDBC classes like Driver Manager, Connection, Statement, and ResultSet.
  2. **Register the JDBC Driver**: Load the database driver using Class.forName() or include it in the class path. For newer versions of JDBC, this step may be optional.
  3. **Open a Connection to the Database**: Use DriverManager.getConnection() to establish a connection to the database using a URL, username, and password.
  4. **Create a Statement**: Use the Connection object to create a Statement object for executing SQL queries.
  5. **Execute SQL Queries**: Use the Statement object to execute SQL commands like SELECT, INSERT, UPDATE, or DELETE.
  6. **Process the Result Set**: If the query returns a result set (e.g., SELECT), use the ResultSet object to iterate through the rows and process the data.
  7. **Close the Connection**: Always close the ResultSet, Statement, and Connection objects to free up resources.

4. Types of JDBC Statements

* Overview of JDBC Statements
  1. Statement: Executes simple SQL queries without parameters.
     + **Create a Connection**: Establish a connection to the database using DriverManager.getConnection().
     + **Create a Statement Object**: Use the Connection object to create a Statement object.
     + **Execute the SQL Query**:
     + Use executeQuery () for SELECT queries. Use executeUpdate () for non-SELECT queries like INSERT, UPDATE, and DELETE.
     + **Process the Result** (for SELECT queries).
     + **Close Resources**: Always close the Statement, ResultSet, and Connection objects to free up resources.
  2. PreparedStatement: Precompiled SQL statements for queries with parameters.
     + **Establish a Database Connection**: Connect to the database using DriverManager.
     + **Create a Prepared Statement Object**:
     + Use the Connection.prepareStatement() method with a parameterized SQL query.
     + Placeholders are denoted by?
     + **Set Parameter Values**:
     + Use methods like setInt, setString, setDate, etc., to set values for each placeholder.
     + **Execute the Query**:
     + Use executeQuery () for SELECT statements or executeUpdate () for INSERT, UPDATE, and DELETE.
     + **Process the Result** (if applicable).
     + **Close the Resources**.
  3. CallableStatement: Used to call stored procedures.
     + **Establish a Database Connection**: Connect to the database using DriverManager.
     + **Prepare a CallableStatement Object**:
       - Use Connection.prepareCall() with the SQL command to call the stored procedure.
       - Syntax: {call procedure name(?, ?, ...)}.
     + **Set Parameters**:
     + Use setInt, setString, etc., for input parameters.
     + Use registerOutParameter () to register output parameters.
     + **Execute the Stored Procedure**:
       - Use execute ().
     + **Retrieve Output Parameters** (if any).
     + **Process the Result Set** (if the stored procedure returns one).
     + **Close the Resources**.
* Differences between Statement, PreparedStatement, and CallableStatement
  1. **CallableStatement:** It is used when you want to use the database stored procedures. CallableStatement can accept runtime input parameters.
  2. **PreparedStatement:** It is used when you want to use SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
  3. **Statement:** It is used for accessing your database. Statement interface cannot accept parameters and useful when you are using static SQL statements at runtime. If you want to run SQL query only once then this interface is preferred over PreparedStatement.

5. JDBC CRUD Operations (Insert, Update, Select, Delete)

* Insert: Adding a new record to the database
  + Adds a new record to the database.
  + Typically uses an INSERT INTO SQL statement.
* Update: Modifying existing records
  + Modifies an existing record in the database.
  + Uses an UPDATE SQL statement with a WHERE clause to target specific rows.
* Select (Read): Retrieving records from the database
  + Retrieves records from the database.
  + Uses a SELECT SQL statement.
  + Results are typically stored in a ResultSet object.
* Delete: Removing records from the database
  + Removes records from the database.
  + Uses a DELETE FROM SQL statement with a WHERE clause.

6. Result Set Interface

* What is ResultSet in JDBC?
  + A ResultSet in JDBC is an object that represents the result of executing a SELECT SQL query. It provides methods to iterate through the rows of a query result and retrieve data from each column in those rows.
* Navigating through ResultSet (first, last, next, previous)

1. **next ()**: Moves the cursor to the next row in the ResultSet.
   * Returns true if there is a next row, and false if you have reached the end of the result set.
2. **previous ()**: Moves the cursor to the previous row.
   * Returns true if there is a previous row, and false if you are at the first row.
3. **first ()**: Moves the cursor to the first row.
   * Returns true if the cursor is successfully moved to the first row, otherwise false.
4. **last ()**: Moves the cursor to the last row.
   * Returns true if the cursor is successfully moved to the last row, otherwise false.
5. **absolute (int row)**: Moves the cursor to a specific row by providing an absolute index.
   * For example, absolute (5) moves the cursor to the 5th row.
   * Returns true if the row exists and the cursor is moved, otherwise false.
6. **beforeFirst () and afterLast ()**: These methods set the cursor before the first row or after the last row respectively.

* Working with ResultSet to retrieve data from SQL queries
  + In JDBC (Java Database Connectivity), the ResultSet object represents the data retrieved from the database after executing a query. It provides various methods to extract the data from the result set and allows you to navigate through the rows returned by the SQL query.

### **Steps for Working with** ResultSet **in JDBC:**

1. **Create a Connection**: Establish a connection to the database using DriverManager.getConnection().
2. **Create a Statement**: Use the Connection object to create a Statement or PreparedStatement.
3. **Execute a Query**: Use Statement.executeQuery() to execute the SQL query and retrieve the results as a ResultSet.
4. **Navigate Through the ResultSet**: Use methods like next (), previous (), first (), last (), etc., to navigate through the rows.
5. **Retrieve Data**: Use methods like getInt (), getString (), getDouble (), etc., to retrieve data from each column of the ResultSet.
6. **Close Resources**: Always close the ResultSet, Statement, and Connection objects to free up resources.

7. Database Metadata

* What is Database Metadata?
  + In JDBC, Database Metadata is an interface that provides information about the database as a whole. It allows you to retrieve metadata (data about data) related to the database, such as the database's structure, capabilities, and limitations. Using Database Metadata, you can get details about tables, columns, indexes, stored procedures, supported features, and other aspects of the database.
* Importance of Database Metadata in JDBC:
  + In JDBC, **Database Metadata** (accessed through the Database Metadata interface) provides information about the database's structure and capabilities. This metadata is crucial for building flexible, database-independent applications. Here’s why it’s important:
    - 1. **Database-Agnostic Programming**: Allows your application to work with multiple databases without changing the code.
      2. **Dynamic Schema Discovery**: Retrieve details about tables, columns, primary/foreign keys, and more, enabling automatic adaptation to schema changes.
      3. **Feature Detection**: Helps check if a database supports specific features like transactions, batch updates, or stored procedures.
      4. **Optimization**: Helps optimize queries based on the database's capabilities.
      5. **Error Prevention**: Ensures compatibility with different databases, reducing the risk of runtime errors.
* Methods provided by Database Metadata (getDatabaseProductName, getTables, etc.)

1. **getDatabaseProductName ()**

* **Description**: Retrieves the name of the database product (e.g., MySQL, Oracle).
* **Return Type**: String
* **Example**:
  + String dBName = dbMetaData.getDatabaseProductName();
  + System.out.println("Database Product Name: " + dbName);

1. **getDatabaseProductVersion ()**

* **Description**: Retrieves the version of the database product.
* **Return Type**: String
* **Example**:
  + String dbVersion = dbMetaData.getDatabaseProductVersion();
  + System.out.println("Database Product Version: " + dbVersion);

1. **getDriverName ()**

* **Description**: Retrieves the name of the JDBC driver.
* **Return Type**: String
* **Example**:
  + String driverName = dbMetaData.getDriverName();
  + System.out.println("JDBC Driver Name: " + driverName);

1. **getTables (String catalog, String schemaPattern, String tableNamePattern, String [] types)**

* **Description**: Retrieves a list of tables in the database that match the given pattern.
* **Return Type**: ResultSet
* **Example**:
  + ResultSet tables = dbMetaData.getTables(null, null, "%", null);
  + while (tables.next()) {
  + String tableName = tables.getString("TABLE\_NAME");
  + System.out.println("Table: " + tableName);
  + }

1. **getColumns (String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)**

* **Description**: Retrieves metadata about the columns in a specified table.
* **Return Type**: ResultSet
* **Example**:
  + ResultSet columns = dbMetaData.getColumns(null, null, "employees", "%");
  + while (columns.next()) {
  + String columnName = columns.getString("COLUMN\_NAME");
  + String columnType = columns.getString("TYPE\_NAME");
  + System.out.println("Column: " + columnName + ", Type: " + columnType);
  + }

1. **getPrimaryKeys (String catalog, String schema, String table)**

* **Description**: Retrieves the primary key for the specified table.
* **Return Type**: ResultSet
* **Example**:
  + ResultSet primaryKeys = dbMetaData.getPrimaryKeys(null, null, "employees");
  + while (primaryKeys.next()) {
  + String pkName = primaryKeys.getString("COLUMN\_NAME");
  + System.out.println("Primary Key: " + pkName);
  + }

1. **getForeignKeys (String catalog, String schema, String table)**

* **Description**: Retrieves information about the foreign keys of a table.
* **Return Type**: ResultSet
* **Example**:
  + ResultSet foreignKeys = dbMetaData.getForeignKeys(null, null, "employees");
  + while (foreignKeys.next()) {
  + String fkName = foreignKeys.getString("FKCOLUMN\_NAME");
  + System.out.println("Foreign Key: " + fkName);
  + }

1. **supportsTransactions ()**

* **Description**: Checks if the database supports transactions.
* **Return Type**: boolean
* **Example**:
  + boolean supportsTx = dbMetaData.supportsTransactions();
  + System.out.println("Supports Transactions: " + supportsTx);

1. **getSchemas ()**

* **Description**: Retrieves all the schemas in the database.
* **Return Type**: ResultSet
* **Example**:
  + ResultSet schemas = dbMetaData.getSchemas();
  + while (schemas.next()) {
  + String schemaName = schemas.getString("TABLE\_SCHEM");
  + System.out.println("Schema: " + schemaName);
  + }

1. **getIndexInfo (String catalog, String schema, String table)**

* **Description**: Retrieves information about the indexes in the specified table.
* **Return Type**: ResultSet
* **Example**:
  + ResultSet indexInfo = dbMetaData.getIndexInfo(null, null, "employees");
  + while (indexInfo.next()) {
  + String indexName = indexInfo.getString("INDEX\_NAME");
  + System.out.println("Index: " + indexName);
  + }

8. ResultSet Metadata

* What is ResultSet Metadata?
  + In JDBC, ResultSetMetaData is an interface that provides metadata about the columns in a ResultSet. It allows you to obtain information about the structure of the data returned by a query, such as column names, data types, column counts, and more. This is particularly useful when you need to work with the result set dynamically, especially when the column details are not known beforehand.
* Importance of ResultSet Metadata in analysing the structure of query results
  + ResultSetMetaData provides important metadata about the structure of the data returned by a SQL query. It allows you to dynamically analyse and interact with the query results, making your code more flexible and database-independent.

**Key Benefits:**

1. **Dynamic Handling**: Helps handle query results without hardcoding column names or types.
2. **Column Information**: Provides details like column names, data types, and sizes (getColumnName (), getColumnType ()).
3. **Database Portability**: Ensures that your application can adapt to different database schemas.
4. **Schema Adaptability**: Allows the program to adjust if the database schema changes, such as added columns.
5. **Error Prevention**: Avoids errors by checking column properties like nullability or read-only status.

* Methods in ResultSetMetaData (getColumnCount, getColumnName, getColumnType)

1. **getColumnCount ()**

* **Description**: Returns the number of columns in the ResultSet.
* **Return Type**: int
* **Example**:
  + int columnCount = rsMetaData.getColumnCount();
  + System.out.println("Number of Columns: " + columnCount);

1. **getColumnName (int column)**

* **Description**: Returns the name of the column at the given index (starting from 1).
* **Return Type**: String
* **Example**:
  + String columnName = rsMetaData.getColumnName(1);
  + System.out.println("Column Name: " + columnName);

1. **getColumnType (int column)**

* **Description**: Returns the SQL type of the column (e.g., Types.INTEGER, Types.VARCHAR).
* **Return Type**: int
* **Example**:
  + int columnType = rsMetaData.getColumnType(1);
  + System.out.println("Column Type: " + columnType);

10. Practical Example 1: Swing GUI for CRUD Operations

* Introduction to Java Swing for GUI development
  + Java Swing is a part of Java's standard library that provides a robust framework for building Graphical User Interfaces (GUIs). It is built on top of the Abstract Window Toolkit (AWT) and offers a more sophisticated set of components for creating cross-platform desktop applications. Swing is part of the Java Foundation Classes (JFC) and is known for its flexibility and ease of use.
* How to integrate Swing components with JDBC for CRUD operations
  + **JFrame**: A top-level container representing a window.
  + **JPanel**: A generic container used to group components.
  + **JLabel**: Used to display a short text or an image.
  + **JButton**: Represents a button that performs an action when clicked.
  + **JTextField**: A single-line text box for user input.
  + **JTextArea**: A multi-line text area for user input.
  + **JComboBox**: A drop-down menu for selecting options.
  + **JTable**: Used to display tabular data

11. Practical Example 2: Callable Statement with IN and OUT Parameters

* What is a CallableStatement?
  + In Java, a CallableStatement is an interface provided by JDBC (Java Database Connectivity) that allows you to execute storedprocedures in a database. Stored procedures are precompiled SQL code stored in the database, and CallableStatement provides a way to call these procedures and handle their inputs and outputs.
* How to call stored procedures using CallableStatement in JDBC
  + To call stored procedures in a database using JDBC, you use the CallableStatement interface. Stored procedures are precompiled SQL code stored in the database, and CallableStatement provides a way to execute these procedures efficiently.