JAVA PROGRAMMING

Chap 10 : Java and Database Programming

Need for Database Programming with Java

- Database programming is an integral part of Java programming because it allows applications to store, retrieve, manipulate, and manage data efficiently. Here are some reasons why database programming is essential in Java:
  - **Data Storage:** Many applications require persistent data storage, where data can be saved and retrieved even after the application is closed or restarted. Databases provide a structured and efficient way to store data.
  - **Data Retrieval:** Java applications often need to retrieve data from databases to present it to users or perform operations on it. Databases allow developers to query and retrieve specific data easily.
  - **Data Manipulation**: Databases enable developers to add, update, and delete data, which is crucial for maintaining the integrity of data and performing operations like CRUD (Create, Read, Update, Delete).
  - **Scalability:** Databases are designed to handle large amounts of data efficiently. Java applications can scale effectively by using databases to store and manage data.
  - **Data Security:** Databases provide security features such as authentication, authorization, and encryption to protect sensitive data. This is crucial for applications that deal with user information or other confidential data.
  - **Data Consistency**: Databases enforce data consistency through constraints and transactions, ensuring that data remains accurate and reliable.
  - **Data Sharing:** Databases enable multiple users or applications to access and share data concurrently while maintaining data integrity.
  - **Reporting and Analysis:** Java applications often need to generate reports or perform complex data analysis. Databases can store and structure data in a way that makes these tasks more efficient.
  - **Persistence Layer:** In many Java applications, a database serves as the persistence layer, separating the application's logic from the data storage concerns. This separation allows for more maintainable and scalable code.

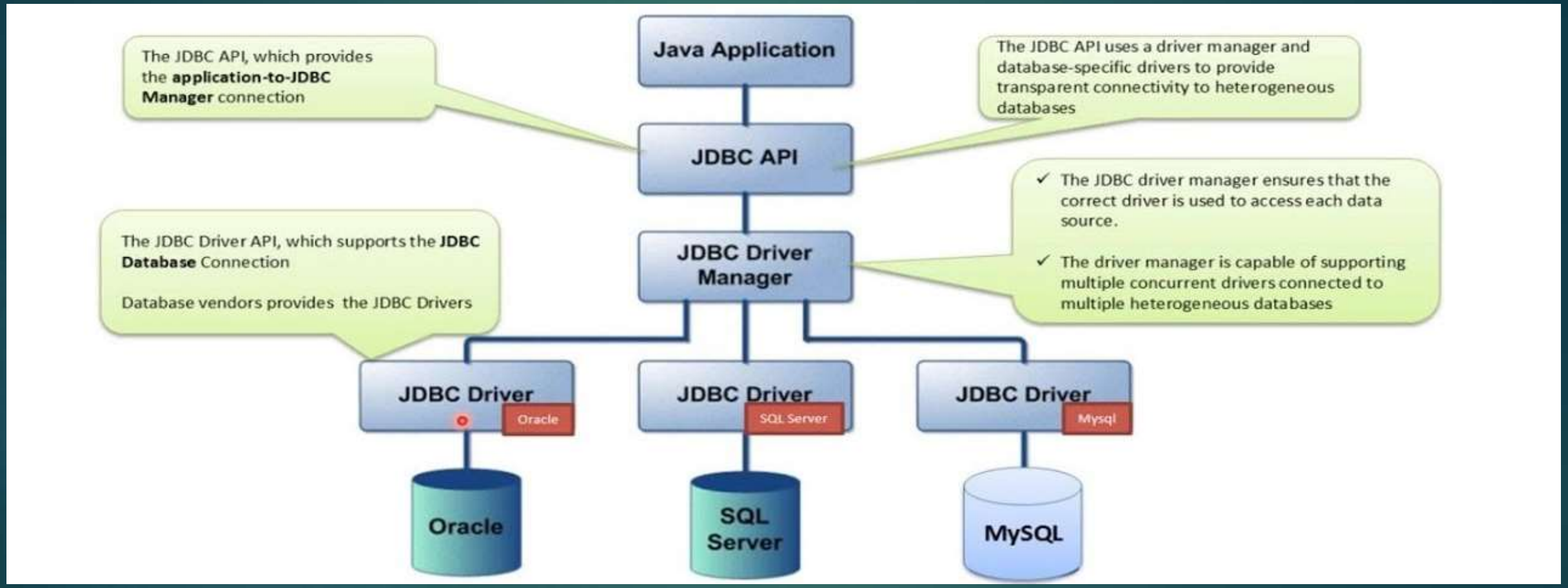
Need for Database Programming with Java

- **Integration:** Java applications can easily integrate with various database management systems (DBMS) such as MySQL, PostgreSQL, Oracle, MongoDB, and others. This flexibility allows developers to choose the database system that best suits their application's requirements.
- To interact with databases in Java, developers typically use Database Connectivity APIs like JDBC (Java Database Connectivity) or Object-Relational Mapping (ORM) frameworks like Hibernate. These tools provide the necessary abstractions and methods to connect to databases, execute queries, and handle database-related tasks seamlessly within Java applications.

JDBC

- JDBC stands for Java Database Connectivity is a Java API(Application Programming Interface) used to interact with databases.
- JDBC is a specification from Sun Microsystems and it is used by Java applications to communicate with relational databases.
- We can use JDBC to execute queries on various databases and perform operations like SELECT, INSERT, UPDATE and DELETE.
- JDBC API helps Java applications interact with different databases like MSSQL, ORACLE, MYSQL, etc.
- It consists of classes and interfaces of JDBC that allow the applications to access databases and send requests made by users to the specified database.
- **Purpose of JDBC**
- Java programming language is used to develop enterprise applications.
- These applications are developed with intention of solving real-life problems and need to interact with databases to store required data and perform operations on it.
- Hence, to interact with databases there is a need for efficient database connectivity, ODBC(Open Database Connectivity) driver is used for the same.
- ODBC is an API introduced by Microsoft and used to interact with databases.
- It can be used by only the windows platform and can be used for any language like C, C++, Java, etc. ODBC is procedural.
- JDBC is an API with classes and interfaces used to interact with databases. It is used only for Java languages and can be used for any platform. JDBC is highly recommended for Java applications as there are no performance or platform dependency problems.

JDBC Architecture



JDBC Architecture

- **Application :** Applications in JDBC architecture are java applications like applets or servlet that communicates with databases.
- **JDBC API :** The JDBC API is an Application Programming Interface used to create Databases. JDBC API uses classes and interfaces to connect with databases. Some of the important classes and interfaces defined in JDBC architecture in java are the DriverManager class, Connection Interface, etc. The primary packages in JDBC are java.sql and javax.sql, which contain classes and interfaces for core database operations.
- **DriverManager :** DriverManager class in the JDBC architecture is used to establish a connection between Java applications and databases. Using the getConnection method of this class a connection is established between the Java application and data sources. It loads the appropriate JDBC driver based on the provided database URL and establishes a connection to the database.
- **JDBC Drivers :** JDBC drivers are used to connecting with data sources. All databases like Oracle, MSSQL, MYSQL, etc. have their drivers, to connect with these databases we need to load their specific drivers. Class is a java class used to load drivers. Class.forName() method is used to load drivers in JDBC architecture. There are four types of JDBC drivers:
  a. Type-1: JDBC-ODBC Bridge Driver (now deprecated)
  b. Type-2: Native-API Driver
  c. Type-3: Network Protocol Driver
  d. Type-4: Thin Driver (also known as the Direct-to-Database Pure Java Driver)
- Each type of driver uses a different mechanism to connect to the database, and the choice of driver depends on the specific database and deployment requirements.

JDBC Architecture

- **Data Sources :** Data Sources in the JDBC architecture are the databases that we can connect using this API. These are the sources where data is stored and used by Java applications. JDBC API helps to connect various databases like Oracle, MYSQL, MSSQL, PostgreSQL, etc.

Types of Drivers

- JDBC (Java Database Connectivity) drivers are platform-specific or database-specific implementations that allow Java applications to connect to and interact with relational databases. There are four main types of JDBC drivers, known as JDBC driver types 1, 2, 3, and 4. Each type has its own characteristics and is suited for different scenarios:

- **Type-1 JDBC Driver (JDBC-ODBC Bridge Driver):**
- Type-1 drivers are also known as JDBC-ODBC Bridge drivers.
- They use the JDBC-ODBC bridge to connect to the database.
- The bridge relies on the ODBC (Open Database Connectivity) driver installed on the system to communicate with the database.
- Type-1 drivers are considered legacy and have been deprecated in favor of other driver types. They are not recommended for new development.

- **Type-2 JDBC Driver (Native-API Driver):**
- Type-2 drivers are known as Native-API drivers.
- They are platform-specific drivers that use a database-specific native API to communicate with the database.
- These drivers provide better performance than Type-1 drivers because they communicate directly with the database without the ODBC layer.
- However, they are still somewhat dependent on the platform and database, which can limit portability.

Types of Drivers

- **Type-3 JDBC Driver (Network Protocol Driver):**
- Type-3 drivers are also called Network Protocol drivers.
- They use a middleware or network protocol to connect to the database server.
- The client-side of the driver translates JDBC calls into a database-independent protocol, which is then transmitted over the network to a server-side component that interacts with the database.
- Type-3 drivers are generally platform-independent and can work with different databases as long as there is a compatible server-side component.

- **Type-4 JDBC Driver (Thin Driver or Direct-to-Database Pure Java Driver):**
- Type-4 drivers are known as Thin drivers.
- They are pure Java drivers that communicate directly with the database using a database-specific protocol.
- Type-4 drivers are highly portable because they don't rely on platform-specific libraries or middleware.
- They are typically the preferred choice for modern Java applications as they offer good performance and platform independence.
- Examples of Type-4 drivers include Oracle Thin Driver, PostgreSQL JDBC Driver, and MySQL Connector/J.

- The choice of JDBC driver type depends on various factors, including the specific database being used, the application's portability requirements, and performance considerations. In general, Type-4 drivers are recommended for new Java applications because they offer good performance and are platform-independent. However, in some legacy or specialized scenarios, other driver types may still be used.

JDBC Components

- **JDBC API :** The JDBC API is an Application Programming Interface used to create Databases. JDBC API uses classes and interfaces to connect with databases. Some of the important classes and interfaces defined in JDBC architecture in java are the DriverManager class, Connection Interface, etc. The primary packages in JDBC are java.sql and javax.sql, which contain classes and interfaces for core database operations.

- **DriverManager :** DriverManager class in the JDBC architecture is used to establish a connection between Java applications and databases. Using the getConnection method of this class a connection is established between the Java application and data sources. It loads the appropriate JDBC driver based on the provided database URL and establishes a connection to the database.

- **JDBC Test Suite :** JDBC Test Suite is used to test operations being performed on databases using JDBC drivers. JDBC Test Suite tests operations such as insertion, updating, and deletion.

- **JDBC-ODBC Bridge Drivers :** As the name suggests JDBC-ODBC Bridge Drivers are used to translate JDBC methods to ODBC function calls. JDBC-ODBC Bridge Drivers are used to connect database drivers to the database. Even after using JDBC for Java Enterprise Applications, we need an ODBC connection for connecting with databases.

  JDBC-ODBC Bridge Drivers are used to bridge the same gap between JDBC and ODBC drivers. The bridge translates the object-oriented JDBC method call to the procedural ODBC function call. It makes use of the sun.jdbc.odbc package. This package includes a native library to access ODBC characteristics.

JDBC Interfaces

- JDBC API uses various interfaces to establish connections between applications and data sources. Some of the popular interfaces in JDBC API are as follows:
- **Driver interface -** The JDBC Driver interface provided implementations of the abstract classes such as java.sql.Connection, Statement, PreparedStatement, Driver, etc. provided by the JDBC API.
- **Connection interface -** The connection interface is used to create a connection with the database. getConnection() method of DriverManager class of the Connection interface is used to get a Connection object.
- **Statement interface -** The Statement interface provides methods to execute SQL queries on the database. executeQuery(), executeUpdate() methods of the Statement interface are used to run SQL queries on the database.
- **ResultSet interface -** ResultSet interface is used to store and display the result obtained by executing a SQL query on the database. executeQuery() method of statement interface returns a resultset object.
- **RowSet interface -** RowSet interface is a component of Java Bean. It is a wrapper of ResultSet and is used to keep data in tabular form.
- **ResultSetMetaData interface -** Metadata means data about data. ResultSetMetaData interface is used to get information about the resultset interface. The object of the ResultSetMetaData interface provides metadata of resultset like number of columns, column name, total records, etc.
- **DatabaseMetaData interface -** DatabaseMetaData interface is used to get information about the database vendor being used. It provides metadata like database product name, database product version, total tables/views in the database, the driver used to connect to the database, etc.

JDBC Classes

- JDBC API uses various classes that implement the above interfaces. Methods of these classes in JDBC API are used to create connections and execute queries on databases. A list of most commonly used class in JDBC API are as follows:
- **DriverManager class -** DriverManager class is a member of the java.sql package. It is used to establish a connection between the database and its driver.
- **Blob class -** A java.sql.Blob is a binary large object that can store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data.
- **Clob class -** The java.sql.Clob interface of the JDBC API represents the CLOB datatype. Since the Clob object in JDBC is implemented using an SQL locator, it holds a logical pointer to the SQL CLOB (not the data).
- **Types class -** Type class defined and store constants that are used to identify generic SQL types also known as JDBC types.

Steps for querying the database with JDBC

- Java applications need to be programmed for interacting with data sources. JDBC Drivers for specific databases are to be loaded in a java application for JDBC support which can be done dynamically at run time. These JDBC drivers communicate with the respective data source.
- Steps to connect a Java program using JDBC API.

1. Import JDBC Packages: Import the necessary JDBC packages at the beginning of your Java program. You'll typically need to import classes from the java.sql package.

2. Load Driver: Load JDBC Driver for specific databases using forName() method of class Class. Syntax: Class.forName("com.mysql.jdbc.Driver")

3. Create Connection: Create a connection with a database using DriverManager class. Database credentials are to be passed while establishing the connection. Syntax: DriverManager.getConnection()

4. Create SQL Query: To manipulate the database we need to create a query using commands like INSERT, UPDATE, DELETE, etc. These queries are created and stored in string format. Syntax: String sql_query = "INSERT INTO Student(name, roll_no) values('ABC','XYZ')"

5. Create SQL Statement: The query we have created is in form of a string. To perform the operations in the string on a database we need to fire that query on the database. To achieve this we need to convert a string object into SQL statements. This can be done using createStatement() and prepareStatement() interfaces.

Syntax: Statement St = con.createStatement();

Steps for querying the database with JDBC

6. Execute Statement: To execute SQL statements on the database we can use two methods depending on which type of query we are executing.

Execute Update: Execute update method is used to execute queries like insert, update, delete, etc. Return type of executeUpdate() method is int. Syntax: int check = st.executeUpdate(sql);

Execute Query: Execute query method is used to execute queries used to display data from the database, such as select. Return type of executeQuery() method is result set. Syntax: Resultset = st.executeUpdate(sql);

7. Closing Statement: After performing operations on the database, it is better to close every interface object to avoid further conflicts. Synatx: con.close();