Lesson:

Database and Java Application Development part 1

PW SKILLS

List of Concepts Involved:

- Introduction to SQL
- DDL and DML Statements
- CRUD operations
- Working with Constraints
- Introduction to JDBC
- Steps followed to create JDBC application
- Need of Statement , PreparedStatement Object
- Connection Pooling, Static vs Dynamic Query
- Introduction to Servlet
- Different ways of creating Servlet
- Scopes in Servlet
- Request Dispatching Mechanism

Introduction to SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

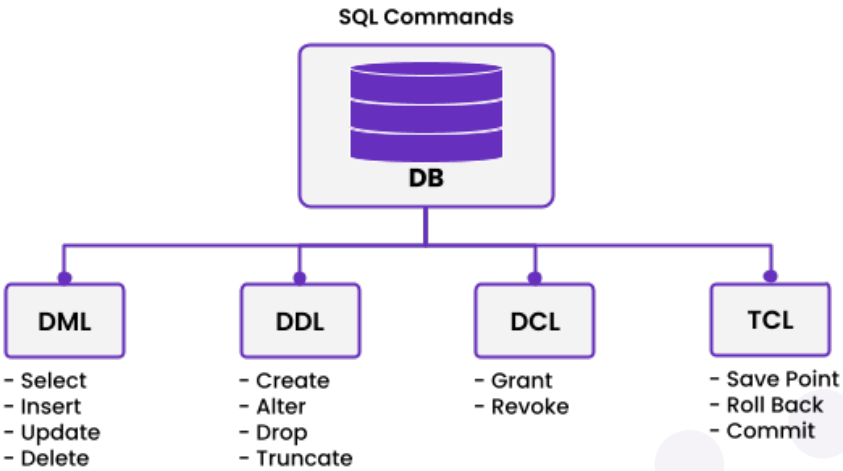
DDL and DML Statements

- DDL commands are SQL commands used to modify or alter the structure of the database. The following is the list of DDL commands in SQL:
- The **CREATE** command creates database objects, tables, and triggers.
- **ALTER** command alters the database structure by adding, deleting, and modifying columns of the already existing tables, like renaming and changing the data type and size of the columns.
- The **DROP** command deletes the defined table with all the table data, associated indexes, constraints, triggers, and permission specifications.
- The **TRUNCATE** command deletes all the data and records from an existing table, including the allocated spaces for the records. Unlike the DROP command, it does not delete the table from the database. It works similarly to the DELETE statement without a WHERE clause.

DML Commands

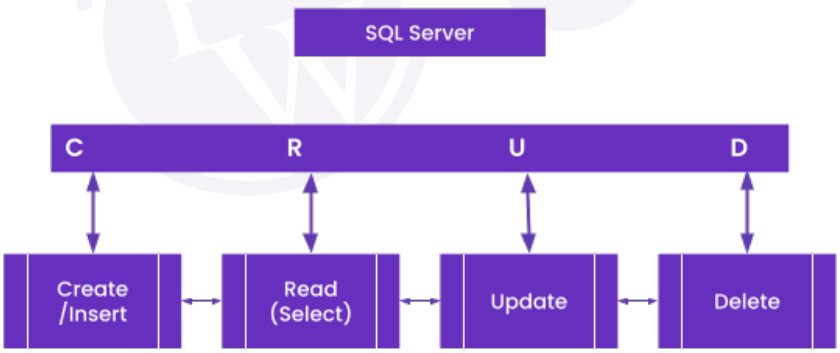
- DML commands are SQL commands that perform operations like storing data in database tables, modifying and deleting existing rows, retrieving data, or updating data.

- The **SELECT** command fetches data or records from one or more tables in the SQL database. The retrieved data gets displayed in a result table known as the result set.
- The **INSERT** command inserts one or more new records into the table in the SQL database.
- The **UPDATE** command updates or changes the existing data or records in a table in the SQL database.
- The **DELETE** command deletes all the existing records and the allocated spaces from a table in the SQL database. We can use the WHERE clause with the AND or OR operators to delete selected rows from the database.



CRUD operations

C: create data,
R: reading the data,
U: update or edit the data,
D: and delete the data.



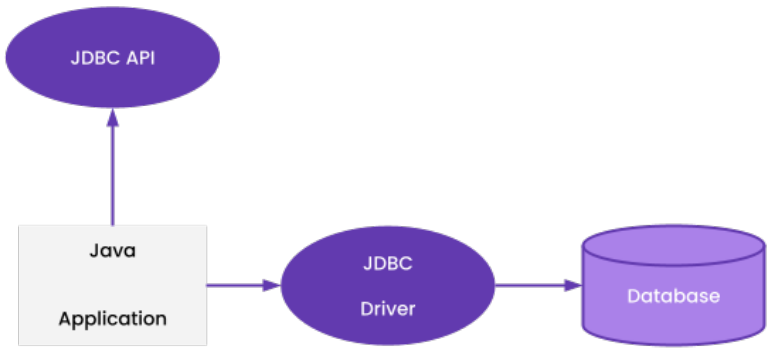
Working with Constraints

1. NOT NULL - Ensures that a column cannot have a NULL value
2. UNIQUE - Ensures that all values in a column are different
3. PRIMARY KEY - A combination of NOT NULL and UNIQUE. Uniquely identifies each row in a table
4. FOREIGN KEY - Prevents actions that would destroy links between tables

5. CHECK - Ensures that the values in a column satisfies a specific condition
6. DEFAULT - Sets a default value for a column if no value is specified

Introduction to JDBC

JDBC is a Java API that is used to connect and execute the query to the database. JDBC API uses JDBC drivers to connect to the database. JDBC API can be used to access tabular data stored into any relational database.



Steps followed to create JDBC application

Steps followed for developing JDBC Application

1. Load and register the Driver
2. Establish the Connection b/w java application and database
3. Create a Statement Object
4. Send and execute the Query
5. Process the result from ResultSet
6. Close the Connection

Types of Statements in JDBC

The statement interface is used to create SQL basic statements in Java; it provides methods to execute queries with the database. There are different types of statements that are used in JDBC as follows:

1. Create Statement
2. Prepared Statement
3. Callable Statement

Create a Statement: From the connection interface, we can create the object for this interface. It is generally used for general–purpose access to databases and is useful while using static SQL statements at runtime.
A Statement object defines several methods to execute different types of SQL statements. In the sample application, the executeQuery() method executes a SELECT statement:
Syntax:

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM emp");
```

Prepared Statement represents a recompiled SQL statement that can be executed many times. This accepts parameterized SQL queries. In this, "?" is used instead of the parameter, one can pass the parameter dynamically by using the methods of PREPARED STATEMENT at run time.
In the people database if there is a need to INSERT some values, SQL statements such as these are used:
INSERT INTO people VALUES ("Ayan",25);
INSERT INTO people VALUES("Kriya",32);

Use Prepared Statements and set the values in the ? holders, setXXX() of a prepared statement is used as shown:

```
String query = "INSERT INTO people(name, age)VALUES(?, ?)";
Statement pstmt = con.prepareStatement(query);
pstmt.setString(1,"Ayan");
ptstmt.setInt(2,25);
// where pstmt is an object name
```

Once the PreparedStatement object is created, there are three ways to execute it:
execute(): This returns a boolean value and executes a static SQL statement that is present in the prepared statement object.
executeQuery(): Returns a ResultSet from the current prepared statement.
executeUpdate(): Returns the number of rows affected by the DML statements such as INSERT, DELETE, and more that is present in the current Prepared Statement.

Callable Statement are stored procedures which are a group of statements that we compile in the database for some task, they are beneficial when we are dealing with multiple tables with complex scenario & rather than sending multiple queries to the database, we can send the required data to the stored procedure & lower the logic executed in the database server itself. The Callable Statement interface provided by JDBC API helps in executing stored **procedures.**
Syntax: To prepare a CallableStatement
CallableStatement cstmt = con.prepareCall("{call Procedure_name(?, ?}");

Once the callable statement object is created
execute() is used to perform the execution of the statement.

Types of Driver available

JDBC Driver is a software component that enables java applications to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1. JDBC-ODBC bridge driver

- The JDBC-ODBC bridge driver uses an 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.

- easy to use.
- can be easily connected to any database.

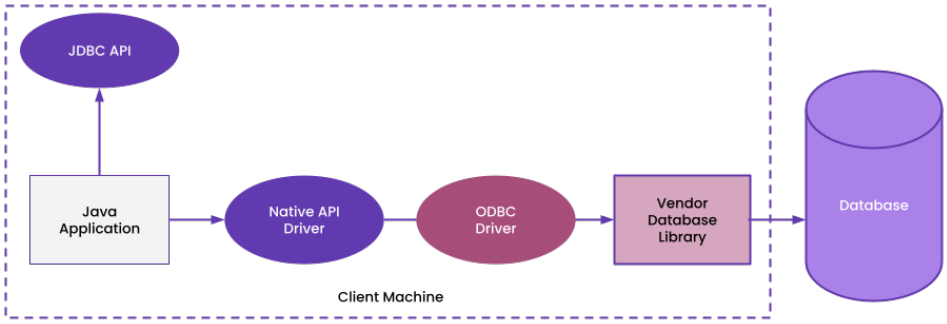


Figure- JDBC-ODBC Bridge Driver

2. 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.
- performance upgraded than JDBC-ODBC bridge driver.

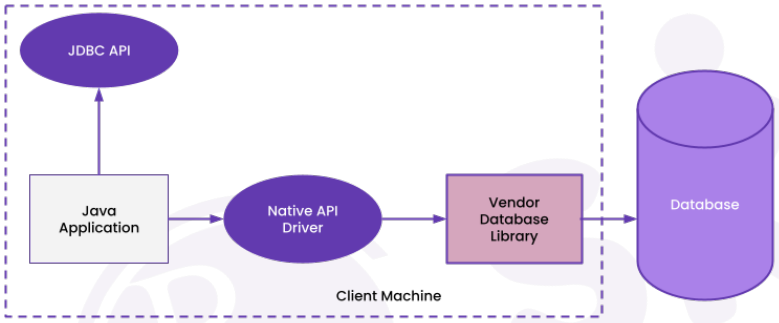


Figure- Native API Driver

3. 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.
- No client side library is required because of the application server that can perform many tasks like auditing, load balancing, logging etc.

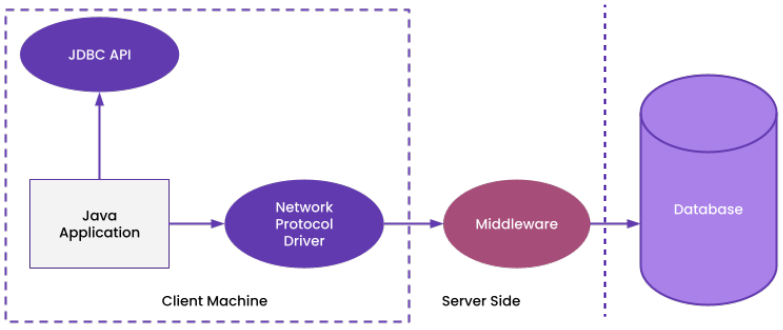


Figure- Network Protocol Driver

4. Thin driver

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as a thin driver. It is fully written in Java language.
- Better performance than all other drivers.
- No software is required at client side or server side.

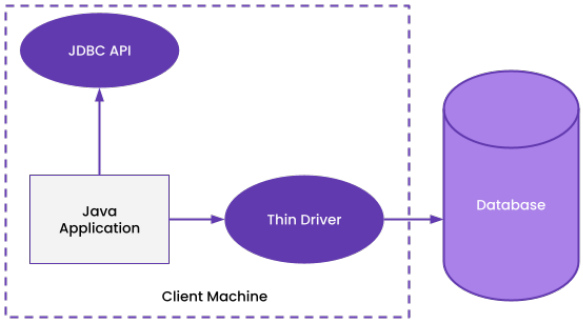


Figure- Thin Driver

Connection Pooling, Static vs Dynamic Query

Connection Pooling: Connections are reused rather than created each time a connection is requested.
Static vs Dynamic Query:

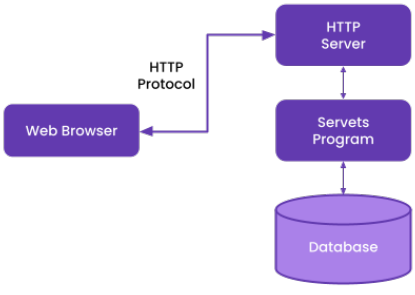
- The sql query without positional parameter(?) is called a static query.
- The sql query with positional parameter(?) is called dynamic query

Servlets

- Servlets are programs that run on a Web or Application server and act as a middle layer between requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.

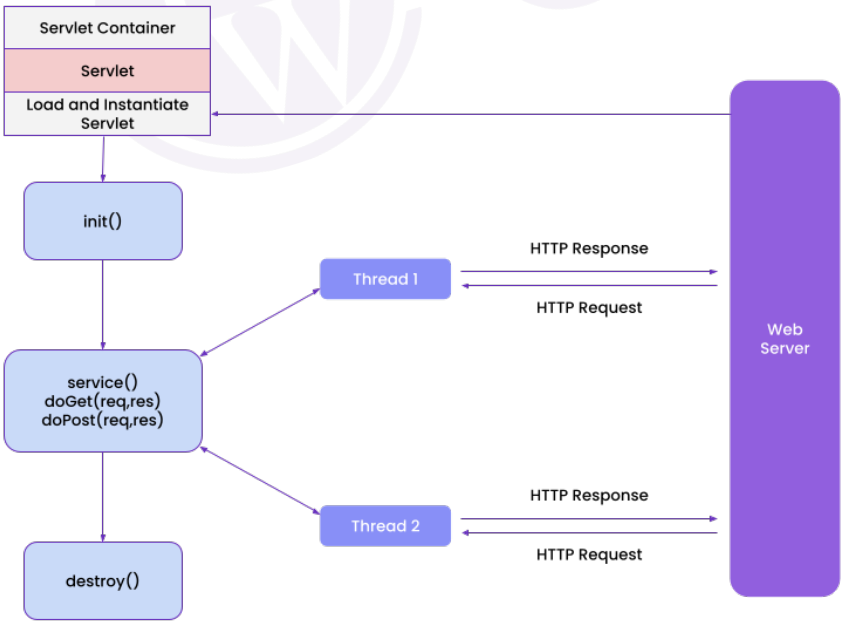


Servlets perform the following tasks:

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.
- Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.
- Servlets can be created using the javax.servlet and javax.servlet.http packages, which are a standard part of Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.
- These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Servlet Life Cycle:

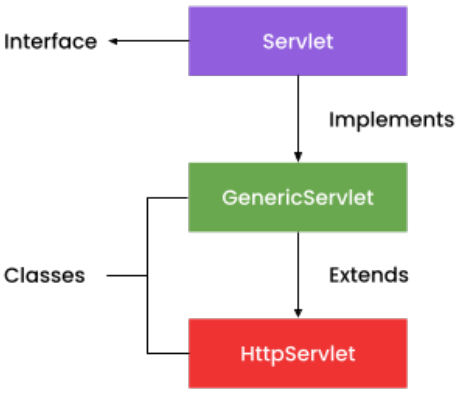
- The Servlet class is loaded first when the Web container receives a new request.
- Then the web container creates an instance of the servlet. This instance is created only once in the whole Servlet Life Cycle.
- The servlet is initialized by the calling init() method.
- service() method is called by the servlet to process the client's request.
- Servlet is destroyed by calling the destroy() method.
- Java Virtual Machine's(JVM) garbage collector clears the destroyed servlet's memory.



Different ways of creating Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface.
2. By inheriting the GenericServlet class.
3. By inheriting the HttpServlet class.



The javax.servlet.Servlet Interface

The javax.servlet.Servlet interface is the basic interface of the Java Servlet API. This provides a standard abstraction for the servlet container to understand the Servlet object created by the user. That means the interface is designed to describe the user-defined Java object to the Servlet container, which encapsulates the application logic to plug into the servlet container for processing the client request. The servlet object must be a subtype of the interface to be managed by the servlet container. The following are the five methods declared in this interface:

1. public abstract void init(ServletConfig)
2. public abstract void service (Service Request, Service Response) throws servlet Exception, IO Exception
3. public abstract void destroy()
4. public abstract void ServletConfiggetServletConfig()
5. public abstract String getServletInfo()

Example: public class myServlet implements Servlet

.....

.....

}

The javax.servlet.GenericServlet Class

- The GenericServlet class implements the servlet interface and, for convenience, the ServletConfig interface.
- Servlet develops typically subclass GenericServlet, or its descendent HttpServlet, unless the servlet needs another class as a parent. (If a servlet does need to subclass another class, the servlet must implement the Servlet interface directly. This would be necessary when, for example, RMI or CORBA objects act as servlets.)
- The GenericServlet class was created to make writing servlets easier. It provides simple versions of the life-cycle and init destroy methods, and of the methods in the ServletConfig interface.
- It also provides a log method, from the ServletContext interface. The servlet writer must override only the service method, which is abstract. Though not required, the servlet implementer should also override the

getServletInfo method, and will want to specialize the init and destroy methods if expensive servlet-wide resources are to be managed. The following are the methods residing in the GenericServlet class.

- public void destroy()
- public String getInitParameter(String name)
- public Enumeration getInitParameterNames()
- public ServletConfiggetServletConfig()
- public ServletContextgetServletContext()
- public String getServletInfo()
- public void init(ServletConfigconfig) throws ServletException
- public void init() throws ServletException
- public void log (Exception e,Stringmsg)
- public void log (String message,Throwable t)
- public abstract· void service (ServletRequest req, ServletResponse res)
- throw Servlet Exception, IOException

Example: public class myServlet extends GenericServlet

.....
.....
}

The javax.servlet.http.HttpServlet Class

- It is an abstract class that simplifies writing HTTP servlets.
- It extends the GenericServlet base class and provides a framework for handling the HTTP protocol. Because it is an abstract class, servlet writers must subclass it and override at least one method.
- The methods normally overridden are;
- doGet, if HTTP GET requests are supported. Overriding the doGet method automatically also provides support for the HEAD and conditional GET operations.
- The getLastModified method should also be overridden, to facilitate caching the HTTP response data.
- This improves performance by enabling smarter conditional GET support.
- doPost, if HTTP POST requests are supported.
- doPut, if HTTP PUT requests are supported.
- doDelete, if HTTP DELETE requests are supported.
- life cycle methods-in it and destroy, if the servlet writer needs to manage resources that are held for the lifetime of the servlet. Servlets that do not manage resources do not need to specialize these methods.
- getServletInfo, To provide descriptive information through a service's administrative interfaces.

Example: public class myServlet extends HttpServlet

.....
.....
}

Scopes in Servlet

There are 3 scopes in the servlet.

1. Request scope.
2. Session scope.
3. Application scope/ ServletContext scope.

Method	Request Scope	Session Scope	Context Scope
Interface	ServletRequest	HttpSession	Context
Object getAttribute(String name)	Returns value of specified name attribute, or null if no attribute with specified name found.	Returns value of bound object with specified name for this session, or null if bound object not found.	Returns value of specified name attribute, or null if no attribute with specified name found.
Enumeration getAttributeNames()	Returns an Enumeration containing names of all the attributes available to this request, or an empty Enumeration if the request has none.	Returns an Enumeration containing attribute names of all objects bound to this session, or an empty Enumeration if the session has none.	Returns an Enumeration containing all attribute names available within this servlet context.
removeAttribute(String name)	Remove attribute with specified name from this request.	Removes bound object with specified name from this session.	Removes attribute with specified name from the servlet context. setAttribute
setAttribute (String name, Object value)	Store attribute with specified name in this request	Binds object to this session using specified name.	Binds an object to a specified name attribute in this servlet context.

Request Dispatching Mechanism

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the ways of servlet collaboration.

Methods of RequestDispatcher interface

There are two methods defined in the RequestDispatcher interface

1. public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException: Includes the content of a resource (servlet, JSP page, or HTML file) in the response.