**JDBC Architecture**

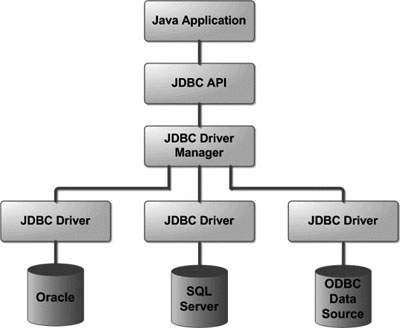
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

* **JDBC API** − This provides the application-to-JDBC Manager connection.
* **JDBC Driver API** − This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



Common JDBC Components

The JDBC API provides the following interfaces and classes −

* **DriverManager** − 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 DriverManager 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.
* **SQLException** − This class handles any errors that occur in a database application.

# JDBC - SQL Syntax

**S**tructured **Q**uery **L**anguage (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.

SQL is supported by almost any database you will likely use, and it allows you to write database code independently of the underlying database.

This chapter gives an overview of SQL, which is a prerequisite to understand JDBC concepts. After going through this chapter, you will be able to Create, **C**reate, **R**ead, **U**pdate, and **D**elete (often referred to as **CRUD** operations) data from a database.

For a detailed understanding on SQL, you can read our [MySQL Tutorial](https://www.tutorialspoint.com/mysql/index.htm).

## Create Database

The CREATE DATABASE statement is used for creating a new database. The syntax is −

SQL> CREATE DATABASE DATABASE\_NAME;

## Example

The following SQL statement creates a Database named EMP −

SQL> CREATE DATABASE EMP;

## Drop Database

The DROP DATABASE statement is used for deleting an existing database. The syntax is −

SQL> DROP DATABASE DATABASE\_NAME;

**Note** − To create or drop a database you should have administrator privilege on your database server. Be careful, deleting a database would loss all the data stored in the database.

## Create Table

The CREATE TABLE statement is used for creating a new table. The syntax is −

SQL> CREATE TABLE table\_name

(

column\_name column\_data\_type,

column\_name column\_data\_type,

column\_name column\_data\_type

...

);

## Example

The following SQL statement creates a table named Employees with four columns −

SQL> CREATE TABLE Employees

(

id INT NOT NULL,

age INT NOT NULL,

first VARCHAR(255),

last VARCHAR(255),

PRIMARY KEY ( id )

);

## Drop Table

The DROP TABLE statement is used for deleting an existing table. The syntax is −

SQL> DROP TABLE table\_name;

## Example

The following SQL statement deletes a table named Employees −

SQL> DROP TABLE Employees;

## INSERT Data

The syntax for INSERT, looks similar to the following, where column1, column2, and so on represents the new data to appear in the respective columns −

SQL> INSERT INTO table\_name VALUES (column1, column2, ...);

## Example

The following SQL INSERT statement inserts a new row in the Employees database created earlier −

SQL> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');

## SELECT Data

The SELECT statement is used to retrieve data from a database. The syntax for SELECT is −

SQL> SELECT column\_name, column\_name, ...

FROM table\_name

WHERE conditions;

The WHERE clause can use the comparison operators such as =, !=, <, >, <=,and >=, as well as the BETWEEN and LIKE operators.

## Example

The following SQL statement selects the age, first and last columns from the Employees table, where id column is 100 −

SQL> SELECT first, last, age

FROM Employees

WHERE id = 100;

The following SQL statement selects the age, first and last columns from the Employees table where *first* column contains *Zara* −

SQL> SELECT first, last, age

FROM Employees

WHERE first LIKE '%Zara%';

## UPDATE Data

The UPDATE statement is used to update data. The syntax for UPDATE is −

SQL> UPDATE table\_name

SET column\_name = value, column\_name = value, ...

WHERE conditions;

The WHERE clause can use the comparison operators such as =, !=, <, >, <=,and >=, as well as the BETWEEN and LIKE operators.

## Example

The following SQL UPDATE statement changes the age column of the employee whose id is 100 −

SQL> UPDATE Employees SET age=20 WHERE id=100;

## DELETE Data

The DELETE statement is used to delete data from tables. The syntax for DELETE is −

SQL> DELETE FROM table\_name WHERE conditions;

The WHERE clause can use the comparison operators such as =, !=, <, >, <=,and >=, as well as the BETWEEN and LIKE operators.

## Example

The following SQL DELETE statement deletes the record of the employee whose id is 100 −

SQL> DELETE FROM Employees WHERE id=100;

# **Java Database Connectivity with 5 Steps**

|  |
| --- |
| There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:   * Register the Driver class * Create connection * Create statement * Execute queries * Close connection |



### **1) Register the driver class**

|  |
| --- |
| The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. |

### **Syntax of forName() method**

1. **public** **static** **void** forName(String className)**throws** ClassNotFoundException

#### **Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.**

### **Example to register the OracleDriver class**

Here, Java program is loading oracle driver to esteblish database connection.

1. Class.forName("oracle.jdbc.driver.OracleDriver");

### **2) Create the connection object**

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with the database. |

### **Syntax of getConnection() method**

1. 1) **public** **static** Connection getConnection(String url)**throws** SQLException
2. 2) **public** **static** Connection getConnection(String url,String name,String password)
3. **throws** SQLException

### **Example to establish connection with the Oracle database**

1. Connection con=DriverManager.getConnection(
2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");

### **3) Create the Statement object**

|  |
| --- |
| The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. |

### **Syntax of createStatement() method**

1. **public** Statement createStatement()**throws** SQLException

### **Example to create the statement object**

1. Statement stmt=con.createStatement();

### **4) Execute the query**

|  |
| --- |
| The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table. |

### **Syntax of executeQuery() method**

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

### **Example to execute query**

1. ResultSet rs=stmt.executeQuery("select \* from emp");
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

### **5) Close the connection object**

|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

### **Syntax of close() method**

1. **public** **void** close()**throws** SQLException

### **Example to close connection**

1. con.close();