JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

* Making a connection to a database.
* Creating SQL or MySQL statements.
* Executing SQL or MySQL queries in the database.
* Viewing & Modifying the resulting records.

## **What is API**

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

Applications of JDBC

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

* Java Applications
* Java Applets
* Java Servlets
* Java ServerPages (JSPs)
* Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

The JDBC 4.0 Packages

The java.sql and javax.sql are the primary packages for JDBC 4.0. This is the latest JDBC version at the time of writing this tutorial. It offers the main classes for interacting with your data sources.

The new features in these packages include changes in the following areas −

* Automatic database driver loading.
* Exception handling improvements.
* Enhanced BLOB/CLOB functionality.
* Connection and statement interface enhancements.
* National character set support.
* SQL ROWID access.
* SQL 2003 XML data type support.
* Annotations.

## **What is JDBC?**

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

* Making a connection to a database.
* Creating SQL or MySQL statements.
* Executing SQL or MySQL queries in the database.
* Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

* Java Applications
* Java Applets
* Java Servlets
* Java ServerPages (JSPs)
* Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

## **Pre-Requisite**

Before moving further, you need to have a good understanding of the following two subjects −

* [Core JAVA Programming](https://www.tutorialspoint.com/java/index.htm)
* [SQL or MySQL Database](https://www.tutorialspoint.com/mysql/index.htm)

## **JDBC Architecture**

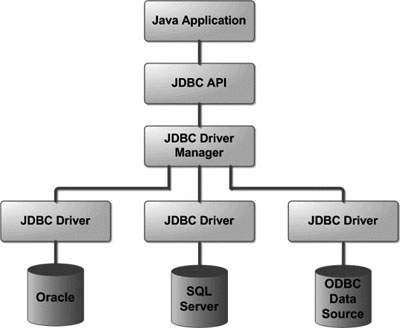
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

* **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.

**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.

## **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;

## **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;

## **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;

## **Create Data Records**

Finally you create few records in Employee table as follows −

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

Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');

Query OK, 1 row affected (0.00 sec)

JDBC - Database Connections

# 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 Driver manager * 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**

Class.forName(“oracle.jdbc.driver.oracleDriver”);

Class.forName(“mysql.jdbcr.Driver”);

Com.mysql.jdbc.driver

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

### **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**

Connection con = Drivermanager.getconn();

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:3306:xe","root","ch");

### **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**

Con.

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)register driver --for name Driver (mysql)

2)create connection-getConn Driver manager-mysql,name,password

3)create statement-crrate st -Conn

4)execute-execu-Statement -Resltset

5)close-

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

### **Example to close connection**

1. con.close();

|  |  |  |
| --- | --- | --- |
| **RDBMS** | **JDBC driver name** | **URL format** |
| MySQL | com.mysql.jdbc.Driver | **jdbc:mysql://**hostname/ databaseName |
| ORACLE | oracle.jdbc.driver.OracleDriver | **jdbc:oracle:thin:@**hostname:port Number:databaseName |
| DB2 | COM.ibm.db2.jdbc.net.DB2Driver | **jdbc:db2:**hostname:port Number/databaseName |
| Sybase | com.sybase.jdbc.SybDriver | **jdbc:sybase:Tds:**hostname: port Number/databaseName |

|  |  |
| --- | --- |
| **Interfaces** | **Recommended Use** |
| Statement | Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. |
| PreparedStatement | Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. |

## **Creating PreparedStatement Object**

PreparedStatement pstmt = null;

try {

String SQL = "Update Employees SET age = ? WHERE id = ?";

pstmt = conn.prepareStatement(SQL);

. . .

}

catch (SQLException e) {

. . .

}

finally {

. . .

}

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost:3306/emp1";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

# JDBC - Create Database Example

**import** java.sql.\*;

**public** **class** JDBCExample {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

) {

String sql = "CREATE DATABASE STUDENT2";

stmt.executeUpdate(sql);

System.***out***.println("Database created successfully...");

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

# JDBC - Create Database Example

**package** com.jdbc;

**import** java.sql.\*;

**public** **class** Jdbccreatedatabase {

**public** **static** **void** main(String[] args) {

**try** {

Class.*forName*("com.mysql.jdbc.Driver");

Connection con=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/", "root","chiranjeevi@12345");

Statement stmt = con.createStatement();

String sql = "CREATE DATABASE Student";

stmt.executeUpdate(sql);

System.***out***.println("Database created successfully...");

}

**catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

}

}

}

# JDBC - Select Database Example

**import** java.sql.\*;

**public** **class** JDBCExample {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/students";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**public** **static** **void** main(String[] args) {

// Open a connection

System.***out***.println("Connecting to a selected database...");

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);) {

System.***out***.println("Connected database successfully...");

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# JDBC - Drop Database Example

**public** **class** JDBCExample {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

) {

String sql = "DROP DATABASE STUDENTS";

stmt.executeUpdate(sql);

System.***out***.println("Database dropped successfully...");

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# JDBC - Create Table Example

## **Required Steps**

The following steps are required to create a new Database using JDBC application −

* **Import the packages** − Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.\** will suffice.
* **Open a connection** − Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with a database server.
* **Execute a query** − Requires using an object of type Statement for building and submitting an SQL statement to create a table in a seleted database.
* **Clean up the environment** − try with resources automatically closes the resources.

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** TestApplication {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/student";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

) {

String sql = "CREATE TABLE REGISTRATION " +

"(id INTEGER not NULL, " +

" first VARCHAR(255), " +

" last VARCHAR(255), " +

" age INTEGER, " +

" PRIMARY KEY ( id ))";

stmt.executeUpdate(sql);

System.***out***.println("Created table in given database...");

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# JDBC - Drop Table Example

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** TestApplication {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/student";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

) {

String sql = "DROP TABLE REGISTRATION";

stmt.executeUpdate(sql);

System.***out***.println("Table deleted in given database...");

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# JDBC - Insert Records Example

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** TestApplication {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/student";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

) {

// Execute a query

System.***out***.println("Inserting records into the table...");

String sql = "INSERT INTO Registration VALUES (100, 'Zara', 'Ali', 18)";

stmt.executeUpdate(sql);

sql = "INSERT INTO Registration VALUES (101, 'Mahnaz', 'Fatma', 25)";

stmt.executeUpdate(sql);

sql = "INSERT INTO Registration VALUES (102, 'Zaid', 'Khan', 30)";

stmt.executeUpdate(sql);

sql = "INSERT INTO Registration VALUES(103, 'Sumit', 'Mittal', 28)";

stmt.executeUpdate(sql);

System.***out***.println("Inserted records into the table...");

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# JDBC - Select Records Example

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** TestApplication {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/student";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**static** **final** String ***QUERY*** = "SELECT id, first, last, age FROM Registration";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(***QUERY***);

) {

**while**(rs.next()){

//Display values

System.***out***.print("ID: " + rs.getInt("id"));

System.***out***.print(", Age: " + rs.getInt("age"));

System.***out***.print(", First: " + rs.getString("first"));

System.***out***.println(", Last: " + rs.getString("last"));

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# JDBC - Update Records Example

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** TestApplication {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/student";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**static** **final** String ***QUERY*** = "SELECT id, first, last, age FROM Registration";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

) {

String sql = "UPDATE Registration " +

"SET age = 30 WHERE id in (100, 101)";

stmt.executeUpdate(sql);

ResultSet rs = stmt.executeQuery(***QUERY***);

**while**(rs.next()){

//Display values

System.***out***.print("ID: " + rs.getInt("id"));

System.***out***.print(", Age: " + rs.getInt("age"));

System.***out***.print(", First: " + rs.getString("first"));

System.***out***.println(", Last: " + rs.getString("last"));

}

rs.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# JDBC - Delete Records Example

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**public** **class** TestApplication {

**static** **final** String ***DB\_URL*** = "jdbc:mysql://localhost/student";

**static** **final** String ***USER*** = "root";

**static** **final** String ***PASS*** = "chiranjeevi@12345";

**static** **final** String ***QUERY*** = "SELECT id, first, last, age FROM Registration";

**public** **static** **void** main(String[] args) {

// Open a connection

**try**(Connection conn = DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

Statement stmt = conn.createStatement();

) {

String sql = "DELETE FROM Registration " +

"WHERE id = 101";

stmt.executeUpdate(sql);

ResultSet rs = stmt.executeQuery(***QUERY***);

**while**(rs.next()){

//Display values

System.***out***.print("ID: " + rs.getInt("id"));

System.***out***.print(", Age: " + rs.getInt("age"));

System.***out***.print(", First: " + rs.getString("first"));

System.***out***.println(", Last: " + rs.getString("last"));

}

rs.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

# }

# PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

1. String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

### **Why use PreparedStatement?**

**Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

#### **ow to get the instance of PreparedStatement?**

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

1. **public** PreparedStatement prepareStatement(String query)**throws** SQLException{}

### **Example of PreparedStatement interface that inserts the record**

**import** java.sql.\*;

**class** InsertPrepare{

**public** **static** **void** main(String args[]){

**try**{

Class.*forName*("com.mysql.jdbc.Driver");

Connection con=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/student", "root","chiranjeevi@12345");

PreparedStatement stmt=con.prepareStatement("insert into registration(id,first) values(?,?)");

stmt.setInt(1,101);//1 specifies the first parameter in the query

stmt.setString(2, "raju") ;

**int** i=stmt.executeUpdate();

System.***out***.println(i+" records inserted");

con.close();

}**catch**(Exception e){ System.***out***.println(e);}

}

# }

My sql connector :

<https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.30>