

The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

* Driver interface
* Connection interface
* Statement interface
* PreparedStatement interface
* CallableStatement interface
* ResultSet interface
* ResultSetMetaData interface
* DatabaseMetaData interface
* RowSet interface

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

## 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.

JDBC Driver

JDBC Driver is a software component that enables java application 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)

# Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

#### NOTE: By default, connection commits the changes after executing queries.

### Commonly used methods of Connection interface:

1. **public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
2. **public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
3. **public void setAutoCommit(boolean status):** is used to set the commit status.By default it is true.
4. **public void commit():** saves the changes made since the previous commit/rollback permanent.
5. **public void rollback():** Drops all changes made since the previous commit/rollback.
6. **public void close():** closes the connection and Releases a JDBC resources immediately.

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

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

2) Create the connection object

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

Syntax of getConnection() method

1) **public** **static** Connection getConnection(String url)**throws** SQLException

2) **public** **static** Connection getConnection(String url,String name,String password)

**throws** SQLException

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

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

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

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

5) Close the connection object

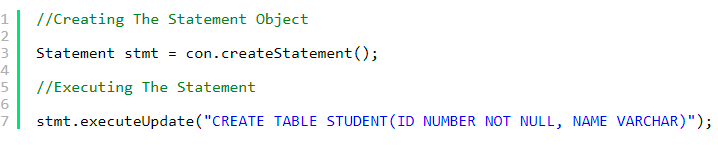
|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

Syntax of close() method

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

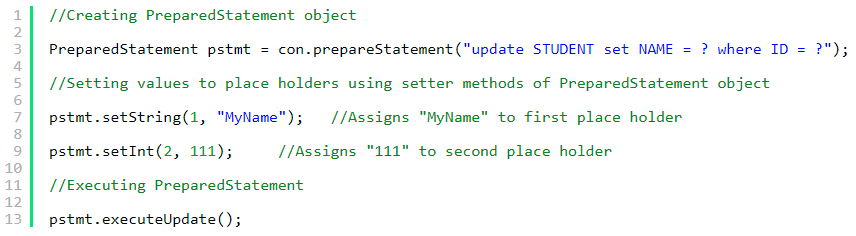
## ****Statement****

Statement interface is used to execute normal SQL queries. You can’t pass the parameters to SQL query at run time using this interface. This interface is preferred over other two interfaces if you are executing a particular SQL query only once. The performance of this interface is also very less compared to other two interfaces. In most of time, Statement interface is used for DDL statements like **CREATE**, **ALTER**, **DROP** etc. For example,



## ****PreparedStatement****

PreparedStatement is used to execute dynamic or parameterized SQL queries. PreparedStatement extends Statement interface. You can pass the parameters to SQL query at run time using this interface. It is recommended to use PreparedStatement if you are executing a particular SQL query multiple times. It gives better performance than Statement interface. Because, PreparedStatement are precompiled and the query plan is created only once irrespective of how many times you are executing that query. This will save lots of time.



## ****CallableStatement****

CallableStatement is used to execute the stored procedures. CallableStatement extends PreparedStatement. Usng CallableStatement, you can pass 3 types of parameters to stored procedures. They are : ***IN*** – used to pass the values to stored procedure, **OUT** – used to hold the result returned by the stored procedure and **IN OUT** – acts as both IN and OUT parameter. Before calling the stored procedure, you must register OUT parameters using **registerOutParameter()** method of CallableStatement. The performance of this interface is higher than the other two interfaces. Because, it calls the stored procedures which are already compiled and stored in the database server.

