# Statement

* Once a connection is obtained we can interact with the database.
* The JDBC framework provides **3 interfaces** and related methods and properties with which we can send SQL or PL/SQL commands to operate on the database.
* These interfaces are:
  1. **Statement interface**
  2. **PreparedStatement interface**

Let us look at them one by one

## Statement (I)

1. Statement is an **Interface** in Java (JDBC)
2. It is present in java.sql package
3. We can obtain a JDBC Statement from a JDBC Connection
4. It is used to execute **static SQL statements** at runtime against an RDBMS.
5. It can be used to execute SQL **DDL** statements, for example data retrieval queries (**SELECT**)
6. It can be used to execute SQL **DML** statements, such as **INSERT**, **UPDATE** and **DELETE**

### Syntax

|  |
| --- |
| Statement stmt = null;  try {  stmt = connection.createStatement( ); // conn is Connection object  . . .  }  catch (SQLException e) {  . . .  }  finally {  if(stmt!=null)  stmt.close();  } |

## CRUD Operations

### Create Operation – INSERT

|  |
| --- |
| Statement statement = connection.createStatement();  String insertQuery = "INSERT INTO students (student\_name, student\_class, student\_fees) values('Kinjal', 1, 5000.0)";  int noOfRowsInserted = statement.executeUpdate(insertQuery); |

### Retrieve Operation – SELECT

|  |
| --- |
| Statement statement = connection.createStatement();  String selectQuery = "SELECT \* FROM students";  ResultSet resultSet = statement.executeQuery(selectQuery);  while (resultSet.next()) {  resultSet.getInt("\_id"); // resultSet.getInt(1);  resultSet.getString("student\_name"); // resultSet.getString(2);  resultSet.getInt("student\_class"); // resultSet.getInt(3);  resultSet.getDouble("student\_fees"); // resultSet.getDouble(4);  } |

### Update – UPDATE

|  |
| --- |
| Statement statement = connection.createStatement();  String updateQuery = "UPDATE students SET student\_class = 12 WHERE student\_class = 11";  int noOfRowsUpdated = statement.executeUpdate(updateQuery); |

### Delete – DELETE

|  |
| --- |
| Statement statement = connection.createStatement();  String deleteQuery = "DELETE FROM students WHERE student\_class = 12";  int noOfRowsDeleted = statement.executeUpdate(deleteQuery); |

### Methods

#### int executeUpdate(String sqlQuery)

This method is used to execute a DML SQL Query (Insert, Updaate and Delete queries)

ResultSet executeQuery(String sqlQuery)

This method is used to retrieve data from the table and returns a result set

The following table provides a summary of each interface's purpose to decide on the interface to use.

|  |  |
| --- | --- |
| **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. |
| CallableStatement | Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters. |

It has the following steps:

### Create Statement

## Step 4. Execute SQL command

### Execute the Query via the Statement

You do so by calling its **executeQuery()** method, passing an SQL statement as parameter.

## Step 5. Collect Information

There are different types of operations that we do with the DB such as insert, update, delete and retrieve. In every case we get different informations.

* In case of insert, update and delete operations, we get number of records inserted, updates or deleted.
* In case of retrieval, we get rows of data in a ResultSet

### ResultSet

1. ResultSet is an Interface in Java (JDBC)
2. It is used to hold records that are returned as a result of a SELECT query
3. We need to travel/traverse/iterate through the ResultSet to get records

### Create ResultSet

|  |
| --- |
| ResultSet resultSet = statement.executeQuery(**sql**); |

### Get the data from ResultSet

1. To iterate the ResultSet you use its **next()** method.
2. The next() method returns **true** if the ResultSet has a next record, and moves to the next record.
3. If there were no more records, next() returns **false**, and you can no longer extract data.

|  |
| --- |
| while(resultSet.next()) {  System.out.println("Id = " + resultSet.getInt(1));  System.out.println("Name = " + resultSet.getString(2));  System.out.println("Class = " + resultSet.getInt(3));  } |

## Step 5. Close the Resources

|  |
| --- |
| resultSet.close();  statement.close();  connection.close(); |