***Difference between exexute,executeUpdate and executeQuery***

1. **boolean execute(String SQL)** : Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
2. **int executeUpdate(String SQL)** : Returns the numbers of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
3. **ResultSet executeQuery(String SQL)** : Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

**Difference between CreateStatement and PreparedStatement**

Differences:  
  
CreateStatement  
(i) Use for general-purpose access to your database.   
(ii) Useful when you are using static SQL statements at runtime because The Statement interface cannot accept parameters.  
  
(iii) Usually suitable  for DDL commands such as Create,alter,drop,etc  
(iv) In terms of efficiency, it is suitable to use Create*Statement* only when we know that we will not need to execute the SQL query multiple times.ex- Statement stmt = con.createStatement();

stmt.executeUpdate("DROP TABLE PRODUCTS IF EXISTS");

where as PreparedStatement Use (i) when you plan to use the SQL statements many times i.e in the context of multiple executions. for eg- suppose a table to require 1000 times a insert same statement,we go for a prepare statment,  
  
(ii) you must use a PreparedStatement object if you want to use large objects like BLOBs or CLOBs.  
(iii) to run dynamic queries beacuse The PreparedStatement interface accepts input parameters at runtime  
ex- String sql="select \* from emp where emp\_id = ?";  
PreparedStatement pStmt = conn.prepareStatement(sql);   
pStmt.setLong(1, profile.getUserId());

here getUserId is given at run time.  
  
Hence we can use *PreparedStatement* to safely provide values to the SQL parameters, through a range of setter methods (i.e. [setInt(int,int)](http://download.oracle.com/javase/6/docs/api/java/sql/PreparedStatement.html#setInt%28int,%20int%29), [setString(int,String)](http://download.oracle.com/javase/6/docs/api/java/sql/PreparedStatement.html#setString%28int,%20java.lang.String%29), etc.).

(iv) The ability to create an incomplete query and supply parameter values at execution time(using ?). This type of query is well suited for filtering queries which may differ in parameter value only:

SELECT firstName FROM employees WHERE salary > 50  
SELECT firstName FROM employees WHERE salary > 200

To create a parametrized prepared statement, use the following syntax:

|  |
| --- |
| // Assume a database connection, conn.  PreparedStatement stmnt = null;  ResultSet rs = null;  try  {  // Create the PreparedStatement, leaving a '?'  // to indicate placement of a parameter.  stmnt = conn.prepareStatement(  "SELECT firstName FROM employees WHERE salary > ?");  // Complete the statement  stmnt.setInt(1, 200);  // Execute the query to obtain the ResultSet  rs = stmnt.executeQuery();  }  catch(Exception ex)  {  System.err.println("Database exception: " + ex);  } |

(v) Prepared is faster because it is precompiled.Most relational databases handles a JDBC / SQL query in four steps:

|  |
| --- |
| 1. Parse the incoming SQL query 2. Compile the SQL query 3. Plan/optimize the data acquisition path 4. Execute the optimized query / acquire and return data |

A Statement will always proceed through the four steps above for each SQL query sent to the database. A PreparedStatement pre-executes steps (1) - (3) in the execution process above. Thus, when creating a PreparedStatement some pre-optimization is performed immediately. The effect is to lessen the load on the database engine at execution time.  
  
(vi)Prepared is an important protection from SQL injection attacks.

Conclusion :  
  
We can perform all operations using both statements but these above described certain  context in which one statement is efficient over another like  
  
For DDL statement use CreateStatement() because these operations used rarely.  
For DMl statements use PreparedStatent(). However if any operation that needs to be performed one or two time during life cycle you can use any one of them.

-----------------------

***Use of different execute() methods in JDBC:-***

* **boolean** execute(*String SQL*) :-  It returns a boolean value true if a ResultSet object can be retrieved; otherwise, it returns false. We can use this method to execute SQL DDL statements.
* **int** executeUpdate(*String SQL*) : - It returns the no of rows updated in db during the execution of the SQL statement. We can use this method to execute SQL queries like, an INSERT, UPDATE, or DELETE Statement.
* **ResultSet** executeQuery(*String SQL*) :- It returns a ResultSet object. We can use this method when we want to execute SELECT queries in db.
* **int[]** executeBatch() :- JDBC allows a program to perform mutiple insertion or updation of data into database at single step. executeBatch() is responsible to execute batch on statement object.

***What will happened when you call destroy() method from the init() method?***

**We can call a destroy() method from init() or service() normally like any other methods.**

**Container will executes the destroy() method but does not unloads the servlet object from the memory.**

**The unloading of servlet can be taken care by container only.**

**It does not make any affect, because destroy method only call by the container when they want to unload the servlet or jsp.**

**By callin destry() within the init() it will call the destroy method, it ll run the code within the destroy() and return back to init(),**

**it does not mean that it will stop the execution process.**