**SQL Assignment 3**

1. Create a function and then call another function from within it. What is this process called?

## Examples of a Function Call from Another Function

Let us see a few different examples, that explain a function call from another function in Python.

**Example 1:**

In this example, the **SumOfSquares()** function calls the **Square()** which returns the square of the number.

* Python3

|  |
| --- |
| # Python code to demonstrate calling the  # function from another function   def Square(X):      # computes the Square of the given number      # and return to the caller function      return (X \* X)   def SumofSquares(Array, n):    # Initialize variable Sum to 0. It stores the      # Total sum of squares of the array of elements      Sum = 0      for i in range(n):    # Square of Array[i] element is stored in SquaredValue          SquaredValue = Square(Array[i])            # Cumulative sum is stored in Sum variable          Sum += SquaredValue      return Sum  # Driver Function  Array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  n = len(Array)   # Return value from the function  # Sum of Squares is stored in Total  Total = SumofSquares(Array, n)  print("Sum of the Square of List of Numbers:", Total) |
|  |

**Output :**

Sum of the Square of List of Numbers: 385

**Example 2:**

In this example, we will call a function from another function within the same class. The class method **Function1()** calls the method **Function2()** from the **Main class**.

* Python3

|  |
| --- |
| '''  Call a function from within another function  in the same class in Python  '''    class Main:    # constructor of Main class      def \_\_init\_\_(self):          # Initialization of the Strings          self.String1 = "Hello"          self.String2 = "World"    def Function1(self):          # calling Function2 Method          self.Function2()          print("Function1:", self.String2)          return    def Function2(self):          print("Function2:", self.String1)          return  # Instance of Class Main  Object = Main()  # Calling Function1  Object.Function1() |

**Output :**

Function2: Hello  
Function1: World

**Example 3:**

In this example, we will call the parent class function from the child class function. The child class inherits the attributes from the parent class.

* Python3

|  |
| --- |
| # Python code to demonstrate calling parent class  # method from the child class method   class Parent:  # constructor of Parent class      def \_\_init\_\_(self):          # Initialization of the Strings          self.String1 = "Hello"          self.String2 = "World"    def Function2(self):          print("Function2:", self.String1)          return  # Child class is inheriting from Parent class  class Child(Parent):    def Function1(self):          # calling Function2 Method in parent class          self.Function2()          print("Function1:", self.String2)          return   ### Instance of Parent class  Object1 = Parent()    ### Instance of Child class  Object2 = Child()  # Calling Function1 using Child class instance  Object2.Function1() |

**Output :**

Function2: Hello  
Function1: World

1. How to inspect the query's execution plan?

# Query Execution Plan in SQL

A **query execution plan** (also known as a **query plan**) is a sequence of steps used by a relational database management system (RDBMS) to access data efficiently when executing a query. The query plan is also referred to as the **SQL Server execution plan**.

The **query optimizer** generates the SQL Server execution plan or query plan. Its goal is to generate an optimal economic query plan. Multiple execution plans are generated by the query processing engine after query execution, and from those generated execution plans, a plan with the best performance is selected.

The execution plans are stored in the **plan cache** memory location, from where they can be reused. Execution plans are available in 3 forms in the SQL server: **XML plans**, **Graphical plans**, and **Text plans**.

Here, we will learn all about **Query-Execution Plans in SQL**, understand the **types of query execution plans**, **how to create** query plans, and at last **how to save query plans** in SQL Server.

## Types of Execution Plan in SQL

There are **two types** of Execution Plans in SQL:

* **Estimated Execution Plan**
* **Actual Execution Plan**

### **1. Actual Execution Plan**

An **actual execution plan is the SQL Server** query plan that is generated after a query has been executed. It contains runtime information, such as actual resource usage metrics and any runtime warnings that occurred during execution.

An actual execution plan displays the actual query execution plan that the SQL Server Database Engine used to execute the queries. You can find the information about actual number of rows processed, resource utilization, and other relevant statistics

### **2. Estimated Execution Plan**

An estimated execution plan is a prediction made by the SQL Server query optimizer regarding the steps it expects to take when executing a query. The estimated execution plan is generated **before** the query is executed.

Estimated execution plans are created during query compilation, before the query is actually executed. These plans are based on statistical information about the database schema, indexes, and data distribution.

An estimated execution plan provides information about the expected sequence of operations. It includes details about the logical and physical operators involved (e.g., scans, joins, sorts).

## Generating and Saving Execution Plans in SQL Server Management Studio

Before and after the execution of the query, the execution plans in SQL Server. Actual and estimated execution plans can be achieved by the given steps:

### Generation of Actual Execution Plans

The actual execution plan can be achieved in the following ways in SQL Server:

1. After completely writing the query, Press Ctrl+M, and the actual execution plan will be generated.
2. Go to the query window and right-click on it, then click on the context menu and select ‘Display Actual Execution Plan’.
3. Or the ‘Display Actual Execution Plan’ icon can be directly selected from the toolbar.

### Generation of Estimated Execution Plans

An estimated execution plan can be achieved using the following ways in SQL Server:

1. After completely writing the query, Press Ctrl+L, and the plan will be generated.
2. Go to the query window and right-click on it, then click on the context menu and select “Display Estimated Execution Plan“.
3. Or the “Display Estimated Execution Plan” icon can be directly selected from the toolbar.

## How to save a Query Execution plan?

One has to save the query plan after interpreting the plan produced by the query. SQL Server Management Studio has an extension of “.sqlplan” for saving the plan in the system.

### **Steps to save an execution plan:**

1. Go to the plan window and right-click.
2. Click on ‘Save Execution Plan As’.
3. Click on the folder or location where you want to save the execution plan, then give the name to the plan and click on ‘Save’.

## Conclusion

In this article, we learn about query-execution plan in SQL. We covered the meaning of SQL Query plans, it’s types and how to create and save these query plans in SQL Server.

RDMS use these query execution plans to follow a sequence of steps while executing a query. An optimized query execution plan save time and money for querying the database.

1. What is the purpose of the MAXDOP and recompiling keywords in SQL queries?

**As Kaboing mentioned, MAXDOP(n) actually controls the number of CPU cores that are being used in the query processor.**

**On a completely idle system, SQL Server will attempt to pull the tables into memory as quickly as possible and join between them in memory. It could be that, in your case, it's best to do this with a single CPU. This might have the same effect as using OPTION (FORCE ORDER) which forces the query optimizer to use the order of joins that you have specified. IN some cases, I have seen OPTION (FORCE PLAN) reduce a query from 26 seconds to 1 second of execution time.**

**Books Online goes on to say that possible values for MAXDOP are:**

**0 - Uses the actual number of available CPUs depending on the current system workload. This is the default value and recommended setting.**

**1 - Suppresses parallel plan generation. The operation will be executed serially.**

**2-64 - Limits the number of processors to the specified value. Fewer processors may be used depending on the current workload. If a value larger than the number of available CPUs is specified, the actual number of available CPUs is used.**

**I'm not sure what the best usage of MAXDOP is, however I would take a guess and say that if you have a table with 8 partitions on it, you would want to specify MAXDOP(8) due to I/O limitations, but I could be wrong.**

1. How to build DDL statements from an existing database table, write steps for it?

# **DDL Commands in SQL**

DDL is an abbreviation of **Data Definition Language**.

The DDL Commands in Structured Query Language are used to create and modify the schema of the database and its objects. The syntax of DDL commands is predefined for describing the data. The commands of Data Definition Language deal with how the data should exist in the database.

**Following are the five DDL commands in SQL:**

1. CREATE Command
2. DROP Command
3. ALTER Command
4. TRUNCATE Command
5. RENAME Command

## **CREATE Command**

CREATE is a DDL command used to create databases, tables, triggers and other database objects.

### **Examples of CREATE Command in SQL**

**Example 1: This example describes how to create a new database using the CREATE DDL command.**

**Syntax to Create a Database:**

**CREATE** **Database** Database\_Name;

Suppose, you want to create a Books database in the SQL database. To do this, you have to write the following DDL Command:

**Create** **Database** Books;

**Example 2: This example describes how to create a new table using the CREATE DDL command.**

**Syntax to create a new table:**

**CREATE** **TABLE** table\_name

(

column\_Name1 data\_type ( **size** **of** the **column** ) ,

column\_Name2 data\_type ( **size** **of** the **column**) ,

column\_Name3 data\_type ( **size** **of** the **column**) ,

...

column\_NameN data\_type ( **size** **of** the **column** )

) ;

Suppose, you want to create a **Student** table with five columns in the SQL database. To do this, you have to write the following DDL command:

**CREATE** **TABLE** Student

(

Roll\_No. **Int** ,

First\_Name **Varchar** (20) ,

Last\_Name **Varchar** (20) ,

Age **Int** ,

Marks **Int** ,

) ;

**Example 3: This example describes how to create a new index using the CREATE DDL command.**

**Syntax to Create a new index:**

**CREATE** **INDEX** Name\_of\_Index **ON** Name\_of\_Table (column\_name\_1 , column\_name\_2 ,  … . , column\_name\_N);

Let's take the Student table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stu\_Id** | **Name** | **Marks** | **City** | **State** |
| 100 | Abhay | 80 | Noida | U.P |
| 101 | Sushil | 75 | Jaipur | Rajasthan |
| 102 | Ankit | 90 | Gurgaon | Haryana |
| 103 | Yogesh | 93 | Lucknow | U.P |

Suppose, you want to create an index on the combination of the **City** and **State** field of the **Student** table. For this, we have to use the following DDL command:

**CREATE** **INDEX** index\_city\_State **ON** Employee (Emp\_City, Emp\_State);

**Example 4: This example describes how to create a trigger in the SQL database using the DDL CREATE command.**

**Syntax to create a trigger:**

**CREATE** **TRIGGER** [trigger\_name]

[ BEFORE | **AFTER** ]

{ **INSERT** | **UPDATE** | **DELETE** }

**ON** [table\_name] ;

## **DROP Command**

DROP is a DDL command used to delete/remove the database objects from the SQL database. We can easily remove the entire table, view, or index from the database using this DDL command.

### **Examples of DROP Command in SQL**

**Example 1: This example describes how to remove a database from the SQL database.**

**Syntax to remove a database:**

**DROP** **DATABASE** Database\_Name;

Suppose, you want to delete the Books database from the SQL database. To do this, you have to write the following DDL command:

**DROP** **DATABASE** Books;

**Example 2: This example describes how to remove the existing table from the SQL database.**

**Syntax to remove a table:**

**DROP** **TABLE** Table\_Name;

Suppose, you want to delete the Student table from the SQL database. To do this, you have to write the following DDL command:

**DROP** **TABLE** Student;

**Example 3: This example describes how to remove the existing index from the SQL database.**

**Syntax to remove an index:**

**DROP** **INDEX** Index\_Name;

Suppose, you want to delete the index\_city from the SQL database. To do this, you have to write the following DDL command:

**DROP** **INDEX** Index\_city;

## **ALTER Command**

ALTER is a DDL command which changes or modifies the existing structure of the database, and it also changes the schema of database objects.

We can also add and drop constraints of the table using the ALTER command.

### **Examples of ALTER Command in SQL**

**Example 1: This example shows how to add a new field to the existing table.**

**Syntax to add a newfield in the table:**

**ALTER** **TABLE** name\_of\_table **ADD** column\_name column\_definition;

Suppose, you want to add the 'Father's\_Name' column in the existing Student table. To do this, you have to write the following DDL command:

**ALTER** **TABLE** Student **ADD** Father's\_Name **Varchar**(60);

**Example 2: This example describes how to remove the existing column from the table.**

**Syntax to remove a column from the table:**

**ALTER** **TABLE** name\_of\_table **DROP** Column\_Name\_1 , column\_Name\_2 , ….., column\_Name\_N;

Suppose, you want to remove the Age and Marks column from the existing Student table. To do this, you have to write the following DDL command:

**ALTER** **TABLE** StudentDROP Age, Marks;

**Example 3: This example describes how to modify the existing column of the existing table.**

**Syntax to modify the column of the table:**

**ALTER** **TABLE** table\_name **MODIFY** ( column\_name column\_datatype(**size**));

Suppose, you want to change the character size of the Last\_Namefield of the Student table. To do this, you have to write the following DDL command:

**ALTER** **TABLE** table\_name **MODIFY** ( Last\_Name **varchar**(25));

## **TRUNCATE Command**

TRUNCATE is another DDL command which deletes or removes all the records from the table.

This command also removes the space allocated for storing the table records.

**Syntax of TRUNCATE command**

**TRUNCATE** **TABLE** Table\_Name;

### **Example**

Suppose, you want to delete the record of the Student table. To do this, you have to write the following TRUNCATE DDL command:

**TRUNCATE** **TABLE** Student;

The above query successfully removed all the records from the student table. Let's verify it by using the following SELECT statement:

**SELECT** \* **FROM** Student;

## **RENAME Command**

RENAME is a DDL command which is used to change the name of the database table.

**Syntax of RENAME command**

RENAME **TABLE** Old\_Table\_Name **TO** New\_Table\_Name;

### **Example**

RENAME **TABLE** Student **TO** Student\_Details ;

This query changes the name of the table from Student to Student\_Details.

1. How to update data in a table using an inner join, write an example?

# **SQL UPDATE with JOIN**

**SQL UPDATE JOIN** means we will update one table using another table and join condition.

Let us take an example of a customer table. I have updated customer table that contains latest customer details from another source system. I want to update the customer table with latest data. In such case, I will perform join between target table and source table using join on customer ID.

Let's see the *syntax* of SQL UPDATE query with JOIN statement.

**UPDATE** customer\_table

**INNER** JOIN

Customer\_table

**ON** customer\_table.rel\_cust\_name = customer\_table.cust\_id

**SET** customer\_table.rel\_cust\_name = customer\_table.cust\_name

### **How to use multiple tables in SQL UPDATE statement with JOIN**

Let's take two tables, table 1 and table 2.

**Create table1**

**CREATE** **TABLE** table1 (column1 **INT**, column2 **INT**, column3 **VARCHAR** (100))

**INSERT** **INTO** table1 (col1, col2, col3)

**SELECT** 1, 11, 'FIRST'

**UNION** ALL

**SELECT** 11,12, 'SECOND'

**UNION** ALL

**SELECT** 21, 13, 'THIRD'

**UNION** ALL

**SELECT** 31, 14, 'FOURTH'

**Create table2**

**CREATE** **TABLE** table2 (column1 **INT**, column2 **INT**, column3 **VARCHAR** (100))

**INSERT** **INTO** table2 (col1, col2, col3)

**SELECT** 1, 21, 'TWO-ONE'

**UNION** ALL

**SELECT** 11, 22, 'TWO-TWO'

**UNION** ALL

**SELECT** 21, 23, 'TWO-THREE'

**UNION** ALL

**SELECT** 31, 24, 'TWO-FOUR'

Now check the content in the table.

**SELECT** \* **FROM** table\_1

1. **SELECT** \* **FROM** table\_2

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 11 | First |
| 2 | 11 | 12 | Second |
| 3 | 21 | **13** | **Third** |
| 4 | 31 | **14** | **Fourth** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 21 | Two-One |
| 2 | 11 | 22 | Two-Two |
| 3 | 21 | **23** | **Two-Three** |
| 4 | 31 | **24** | **Two-Four** |

Our requirement is that we have table 2 which has two rows where Col 1 is 21 and 31. We want to update the value from table 2 to table 1 for the rows where Col 1 is 21 and 31.

We want to also update the values of Col 2 and Col 3 only.

The most easiest and common way is to use join clause in the update statement and use multiple tables in the update statement.

**UPDATE** **table** 1

**SET** Col 2 = t2.Col2,

Col 3 = t2.Col3

**FROM** table1 t1

**INNER** JOIN **table** 2 t2 **ON** t1.Col1 = t2.col1

**WHERE** t1.Col1 IN (21,31)

Check the content of the table

SELECT FROM table 1

SELECT FROM table 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 11 | First |
| 2 | 11 | 12 | Second |
| 3 | 21 | **23** | **Two-Three** |
| 4 | 31 | **24** | **Two-Four** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 21 | First |
| 2 | 11 | 22 | Second |
| 3 | 21 | **23** | **Two-Three** |
| 4 | 31 | **24** | **Two-Four** |

Here we can see that using join clause in update statement. We have merged two tables by the use of join clause.

1. Differentiate between truncate, delete, and drop with a suitable example.

# Difference between DELETE, DROP and TRUNCATE

### **1. DELETE :**

Basically, it is a [Data Manipulation Language Command (DML)](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/). It is used to delete one or more tuples of a table. With the help of the “DELETE” command, we can either delete all the rows in one go or can delete rows one by one. i.e., we can use it as per the requirement or the condition using the Where clause. It is comparatively slower than the TRUNCATE command. The TRUNCATE command does not remove the structure of the table.

* **SYNTAX –**   
  If we want to delete all the rows of the table:

DELETE from;

* **SYNTAX –**   
  If we want to delete the row of the table as per the condition then we use the WHERE clause,

DELETE FROM table\_name WHERE condition;

**Note –** Here we can use the “ROLLBACK” command to restore the tuple because it does not auto-commit.

### **2. DROP :**

It is a Data Definition Language Command (DDL). It is used to drop the whole table. With the help of the “DROP” command we can drop (delete) the whole structure in one go i.e. it removes the named elements of the schema. By using this command the existence of the whole table is finished or say lost.

* **SYNTAX –**   
  If we want to drop the table:

DROP table <table\_name>;

**Note –** Here we can’t restore the table by using the “ROLLBACK” command because it auto commits.

### **3. TRUNCATE :**

It is also a Data Definition Language Command (DDL). It is used to delete all the rows of a relation (table) in one go. With the help of the “TRUNCATE” command, we can’t delete the single row as here WHERE clause is not used. By using this command the existence of all the rows of the table is lost. It is comparatively faster than the delete command as it deletes all the rows fastly.

* **SYNTAX –**   
  If we want to use truncate :

TRUNCATE table <table\_name>;

**Note –** Here we can’t restore the tuples of the table by using the “ROLLBACK” command.

## Comparision Between Delete Drop and Truncate

| **Parameter** | **Delete** | **Drop** | **Truncate** |
| --- | --- | --- | --- |
| **Language** | Data Manipulation Language Command (DML) | Data Definition Language Command (DDL) | Data Definition Language Command (DDL) |
| **Purpose** | Used to delete content in rows of a table. | Used to delete entire content of table along with the table structure. | Used to delete entire content of table leaving the table structure. |
| **Syntax** | DELETE from;(to delete all the rows of the table) DELETE FROM table\_name WHERE condition; (to delete the row of the table as per the condition) | DROP table <table\_name>; | TRUNCATE table <table\_name>; |
| **ROLLBACK** | Can be Rollback | Cannot be Rollback | Cannot be Rollback |
| **Data** | Removes specific rows depending upon condition | Removes the entire data immediately. | Removes all rows |
| **Efficiency** | Less efficient for large tables as we have to manually specify each and every condition. | Efficiency depends on the size of the object which is being dropped. | More efficient for large tables as we are removing all the data in one step. |