LAB # 4

CREATING AND ALTERING TABLES

OBJECTIVE

Creating and altering tables

THEORY

Creating a Table

When you started this section, you looked at the standard CREATE syntax of:

CREATE <*object type*> <*object name*>

And then you moved on to a more specific start (indeed, it's the first line of the statement that creates the table) and created a table called Customers:

SQL PRIMARY KEY Constraint on CREATE TABLE

```
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

To DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

```
ALTER TABLE Persons DROP CONSTRAINT pk_PersonID
```

SQL FOREIGN KEY Constraint on CREATE TABLE

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
```

```
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
)
```

To DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE Orders DROP CONSTRAINT fk_PerOrders
```

Adding Data with the *INSERT* Statement

We can add a new record or row into a table by using the INSERT statement.

SYNTAX

```
INSERT [INTO] <table> [(column_list)]
VALUES (data_values)
```

Where:

Table is the name of the table

Column is the name of the column in the table to populate

Value is the corresponding value for the column

INTO is optional and columns list is also optional and may be omitted if you list values in the default order of the columns in the table.

1. Type the following query.

```
insert into department(dno, dname,dloc)
values (50,'Development','Karachi')
```

This result could also have been achieved by using this query if the column order is default:

```
insert into department values (50,'Development','Karachi')
```

INSERTING NULL VALUES IN ROWS

Implicit Method: *Omit the Column Name From the Column List*

```
insert into department(dno,dname) values(60,'MIS')
```

***Explicit Method*:** Specify the NULL keyword or empty string (‘ ‘) in the values list.

```
insert into department values(70,'Finance',NULL)
```

INSERTING SPECIAL VALUES USING SQL FUNCTIONS

Considering the following query:

```
insert into employee(eno,ename,job,head,hiredate,sal,comm,dno)
values(601,'Shahnawaz','Salesman',125,GETDATE(),20000,1500,10)
```

Here an employee names Shahnawaz has been added to the employee table, it uses the GETDATE() function for current date and time. To confirm the addition of new employee data into the employee’s table, type this query into SSMS:

```
select * from employee where eno=601
```

INSERTING SPECIFIC DATE VALUES

To insert a row or record with some specific date, consider the following query:

```
insert into employee(eno,ename,job,head,hiredate,sal,comm,dno)
values(602,'Iqbal','Salesman',125,'02/11/2009',20000,2500,10)
```

Verify your insertion by using SELECT statement.

INSERTING VALUES BY USING SUBSTITUTION VARIABLE

You can also declare and set different variables in a T-SQL which can be passed to an INSERT statement. Consider the following example.

```
DECLARE @dept_number int
set @dept_number=90
DECLARE @dept_name nvarchar(50)
set @dept_name='Training'
DECLARE @dept_location nvarchar(50)
set @dept_location='Karachi'
```

Here we have declared three variables @dept_number, @dept_name, and @dept_location with data types as int, nvarchar (50) and nvarchar (50) respectively. Before using them into our INSERT statement we have set their values accordingly as 90, ‘Training’, and ‘Karachi’. Now we can use them into an insert statement and run them concurrently/simultaneously.

```
DECLARE @dept_number int
set @dept_number=90
DECLARE @dept_name nvarchar(50)
set @dept_name='Training'
DECLARE @dept_location nvarchar(50)
set @dept_location='Karachi'

insert into department values(@dept_number,@dept_name,@dept_location)
```

The above example is treated as a script that is you are running the entire script or nothing at all. They are all working in unison to accomplish one task—to insert one record into the database.
Check it existence by running the following query:

```
select * from department
```

COPYING ROWS FROM ANOTHER TABLE

You can use the INSERT statement to add rows to a table whereas the values come from existing tables. We do not use values clause, instead use a subquery (which we may discuss later). You should match the number of columns in the INSERT clause to those in the subquery.

Syntax

```
INSERT INTO table(column_list)
        Subquery
```

Where: table is the table name where insert statement is sought for
Column_list name of the columns in the table to be populated
Subquery subquery that returns values into the table

Consider the following query:

```
insert into bonus(ename,job,sal,comm)
        select ename,job,sal,comm
        from employee
        where job='Manager'
```

Here Bonus table has been populated by a SELECT statement from Employee table using the INSERT statement. Check the Bonus table by using this query.

```
select * from bonus
```

Updating/Changing Data With The *UPDATE* Statement

You can update an existing record or row in a table by using the UPDATE statement.

Syntax

UPDATE <table name>
SET <column> = <value>
[WHERE condition

Where: Table is the name of the table
Column is the name of the column(s) in the table to updated
Value is the corresponding value for the column

Updating Rows in a Table

The update statement can modify rows specified by the WHERE clause condition. Omitting the WHERE will result in updating all the records in the table.

a. Consider the following query: Specific row(s) can be updated while using the WHERE clause.

```
update employee set dno=10 where eno=259
```

This query updates the department number information in the employee table from 30 to 10.

i. Consider the following query: All rows in the table will be updated, if we omit the WHERE clause.

```
update temp set Dno=20
```

Temp table has the same data as of the Employee table. You will be provided this in the lab.

Updating with Multiple Columns Subquery

Multiple column sub-query can be used to be implemented in the SET clause to update a row.

Syntax

```
UPDATE table
SET (column, column,) =
        (SELECT column, column, …
        FROM table
WHERE  condition
```

Consider the following query:

```
update employee
set job=(Select job from employee Where eo=231),
   dno=(Select dno FROM employee Where eno=231)
where eno=760
```

Updating Rows Based on another Table

You can use subqueries to update records of a table. Consider the following example:

```
update temp
set dno=(select dno from employee where eno=104),
   job=(select job from employee where eno=104)
```

This changes the department number of all employees in the temp table with that of the employee number 104 in the Employee table and vice versa his job title.

Removing Data With The *DELETE* Statement

You can delete records of table by the DELETE statement.

Syntax

```
DELETE      FROM        table
WHERE               condition
```

Here FROM keyword is optional. Always remember that all the rows of the table will be deleted if not mentioning the WHERE clause.

Deleting Rows from a Table

a. Consider the following query: Specific row(s) can be deleted while using the WHERE clause.

```
delete from department where dname='development'
```

b. Consider the following query: All rows in the table will be deleted, if we omit the WHERE clause.

```
delete from dept
```

Dept table will be provided to you in lab.

Deleting Rows Based on Another Table

You can use subqueries for deleting rows from a table based on values of some other table. Consider the following example:

```
delete from temp
where dno=(select dno from department where dname='Research')
```

All records of temp table will vanish.