**DROP, TRUNCATE and Delete , Alter, CTE's**

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

-- The DROP TABLE statement is used to drop an existing table in a database.

--The following SQL statement drops the existing table "Shippers":

select \* from PRACTICE\_DB.SQL.shippers;

DROP TABLE PRACTICE\_DB.SQL.SHIPPERS;

UNDROP TABLE PRACTICE\_DB.SQL.SHIPPERS;

--SQL TRUNCATE TABLE

--The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

TRUNCATE TABLE PRACTICE\_DB.SQL.SHIPPERS; -- makes table empty at one go.

DELETE FROM PRACTICE\_DB.SQL.SHIPPERS ; -- record by record

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

-- ALTER TABLE :-

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

/\*

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

ALTER TABLE table\_name

ADD column\_name datatype;

The following SQL adds an "Email" column to the "Customers" table:

\*/

-- create or replace TABLE CUSTOMERS (

-- CUSTOMERID NUMBER(38,0),

-- CUSTOMERNAME VARCHAR(16777216),

-- CONTACTNAME VARCHAR(16777216),

-- ADDRESS VARCHAR(16777216),

-- CITY VARCHAR(16777216),

-- POSTALCODE VARCHAR(16777216),

-- COUNTRY VARCHAR(16777216),

-- EMAIL VARCHAR(255)

-- );

select \* from customers;

select get\_ddl('table','practice\_db.sql.customers');

ALTER TABLE Customers

ADD Email varchar(255);

--ALTER TABLE - DROP COLUMN

--To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name

DROP COLUMN column\_name;

--The following SQL deletes the "Email" column from the "Customers" table:

ALTER TABLE Customers

DROP COLUMN Number2 number;

--ALTER TABLE - RENAME COLUMN

--To rename a column in a table, use the following syntax:

ALTER TABLE table\_name

RENAME COLUMN old\_name to new\_name;

--To rename a column in a table in SQL Server, use the following syntax:

--EXEC sp\_rename 'table\_name.old\_name', 'new\_name', 'COLUMN';

--ALTER TABLE - ALTER/MODIFY DATATYPE

--To change the data type of a column in a table, use the following syntax:

ALTER TABLE Customers

ADD COLUMN Number2 number(

);

ALTER TABLE Customers

MODIFY COLUMN postalcode decimal(38,0);

ALTER TABLE Customers

MODIFY COLUMN customerid decimal(38,0);

ALTER TABLE table\_name

MODIFY column\_name datatype;

-- DROP COLUMN Example

ALTER TABLE Persons

DROP COLUMN DateOfBirth;

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

-- CTE's:-

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

/\* CTEs (Common Table Expressions)

See also:

WITH

What is a CTE?

A CTE (common table expression) is a named subquery defined in a WITH clause. You can think of the CTE as a temporary view for use in the statement that defines the CTE. The CTE defines the temporary view’s name, an optional list of column names, and a query expression (i.e. a SELECT statement). The result of the query expression is effectively a table. Each column of that table corresponds to a column in the (optional) list of column names.

The following code is an example of a query that uses a CTE:

WITH

my\_cte (cte\_col\_1, cte\_col\_2) AS (

SELECT col\_1, col\_2

FROM ...

)

SELECT ... FROM my\_cte;

In the example above, the CTE starts on the line containing my\_cte (cte\_col\_1, cte\_col\_2) AS (, and ends on the line containing ).

Avoid choosing CTE names that match the following:

SQL function names

Tables, views, or materialized views. If a query defines a CTE with a particular name, the CTE takes precedence over tables, etc.

A CTE can be recursive or non-recursive. A recursive CTE is a CTE that references itself. A recursive CTE can join a table to itself as many times as necessary to process hierarchical data in the table.

CTEs increase modularity and simplify maintenance.

Recursive CTEs and Hierarchical Data

Recursive CTEs enable you to process hierarchical data, such as a parts explosion (component, sub-components) or a management hierarchy (manager, employees). For more information about hierarchical data, and other ways to query hierarchical data, see Querying Hierarchical Data.

A recursive CTE allows you to join all the levels of a hierarchy without knowing in advance how many levels there are.

Overview of Recursive CTE Syntax

This section provides an overview of the syntax and how the syntax relates to the way that the recursion works:

WITH [ RECURSIVE ] <cte\_name> AS

(

<anchor\_clause> UNION ALL <recursive\_clause>

)

SELECT ... FROM ...;

\*/

WITH customers\_data as (

select \* from PRACTICE\_DB.SQL.CUSTOMER\_COPY

),

employees\_data as (

select \* from PRACTICE\_DB.SQL.EMPLOYEES

)

,

customers\_employees as(

select \* from customers\_data c

join employees\_data e on e.employeeid = c.customerid

)

select \* from customers\_employees;