CMP020L012A	Data Analytics	2022/23

Lab Manual (DDL and DML Operations)
Week-2

DDL Operations:

There are a few considerations to remember while defining any database object (database "table"):

- The name of the table must begin with a "letter"
- The name MUST BE 1-30 characters long
- The name MUST contain only A–Z, a–z, 0–9, _, $, and #
- The name MUST NOT duplicate the name of another object owned by the same user
- The name MUST NOT be a database server reserved word; e.g. "Create" is a reserved word so, it cannot be defined as a name

As a user if the access to "Create" any object is given and there's enough storage area available to perform this task, then creating a new object can be successfully done.

CREATE TABLE [*schema*.]*table*
(*column datatype* [DEFAULT *expr*][, ...]);

Here, the table name must be specified along with the column names, column data type and the size of the column. For example, to create table "dept" with attributes: deptno, dname and location the SQL be like:

SQL> CREATE TABLE dept (deptno NUMBER (2),
dname VARCHAR (14),
loc VARCHAR (13));
Table created.

How do we know if “dept” table has already been created!

SQL> DESCRIBE dept;

```
 Name                          Null?    Type
 ----------------------------- -------- ---------
 DEPTNO                                 NUMBER (2)
 DNAME                                  VARCHAR (14)
 LOC                                    VARCHAR (13)
```

The already created table can also be “Altered” if required:

- To Add a new column to the table
- To Modify the existing Column(s)
- Define default value for the column(s)

ALTER TABLE *table*
ADD (*column datatype* [DEFAULT *expr*]
[, *column datatype*] ...);
OR,

ALTER TABLE *table*
MODIFY (*column datatype* [DEFAULT *expr*]
[, *column datatype*] ...);

Now, let’s say if alter is necessary to add a column “job” in the existing table “dept” then the SQL will be like:

SQL> ALTER TABLE dept
ADD (address VARCHAR (9));

As a result, the table will look like:

```
 Name                          Null?       Type
 ----------------------------- --------    ---------
  DEPTNO                                   NUMBER (2)
  DNAME                                    VARCHAR (14)
  LOC                                      VARCHAR (13)
  ADDRESS                                  VARCHAR (9)
```

Now, if any modification of any column is required then that can be done too, for example, to change the data type, size and/or, default value, we can proceed with those alteration(s) on the existing database table.

SQL> ALTER TABLE dept
MODIFY (address VARCHAR (20));

```
 Name                          Null?       Type
 ----------------------------- --------    ---------
  DEPTNO                                   NUMBER (2)
  DNAME                                    VARCHAR (14)
  LOC                                      VARCHAR (13)
  ADDRESS                                  VARCHAR (20)
```

Alteration of a table alters the definition the table to some extent but to discard the entire table from the database it needs to be dropped explicitly. After the drop operation on any table:

- All data and the structure in the table is deleted
- Any pending transactions are committed
- All indexes of the tables are dropped

SQL> DROP TABLE dept;
Table dropped.

In order to change the name of the object (table) , view, or sequence "RENAME" statement is executed. Sometimes the keyword "Change" would execute instead.

SQL> RENAME TABLE dept to Department;
Table renamed.

Another type of definition, called "TRUNCATE" is used to remove all rows from the table eventually that releases the storage space used by that table.

SQL> TRUNCATE TABLE Department;
Table truncated.

Summary

Statement	Description
CREATE TABLE	Creates table
ALTER TABLE	Modifies the Structure
DROP TABLE	Removes the rows and the table structure
RENAME	Changes the name of a table, view, sequence, or synonym
TRUNCATE	Removes all rows from a table and releases the storage space

*****DDL makes changes on any object, means the result of the change, impacts for all the rows present within the object.**

Data types that are widely used in Relational Database System

Data Types	Description
CHAR	A FIXED length string (can contain letters, numbers, and special characters).
VARCHAR	A VARIABLE length string (can contain letters, numbers, and special characters)
BINARY	Like CHAR(), but stores binary byte strings
FLOAT	Floating precision number data
BOOLEAN/BOOL	Zero is considered as false, nonzero values are considered as true
INT/ INTEGER	A medium integer.
BIGINT	A large integer.
DOUBLE (S, P)	A normal-size floating point number. The total number of digits is specified in *size*.
DATE	A date. Format: YYYY-MM-DD
DATETIME	A date and time combination. Format: YYYY-MM-DD hh:mm:ss
TIMESTAMP	TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
YEAR	A year in four-digit format.
DECIMAL (S, P)	An exact fixed-point number. The total number of digits is specified in *size*.
IMAGE	Variable width binary string
TEXT	Variable width character string
ENUM (val1,val2,val3…)	A string object that can have only one value, chosen from a list of possible values.

Constraints in DDL Operations:

Database constraints work at the table level, which enforce rules whenever a row is inserted, updated, or deleted from that table. The constraint types that will be used are:

Primary Key
Foreign Key
NOT NULL
Unique Key

The constraints can be defined,

- At the same time when the **table is created**, or
- On the existing table **by alteration**

Constraints are defined at the column level or table level and can be viewed in the data dictionary.

Primary Key constraint helps to identify each record uniquely in a database table. However, this constraint restricts the value under this attribute are "unique". It also assures the attribute is not NULL. For example, in a employee table employee's ID can be considered as a primary key because:
Each employee has ONLY ONE employee ID and this ID cannot be same with other employees, hence any specific employee's record can be identified by this attribute.

***A table has ONLY ONE Primary Key attribute at the same time.
To add constraint while defining table we perform the SQL as follows:

CREATE TABLE [*schema*.]*table*
(*column datatype* [DEFAULT *expr*]
[*column_constraint*],
...
[*table_constraint*][,...]);

For example,

```
SQL>CREATE TABLE Employee(
        empno  INT(4),
        ename  VARCHAR(10),
        salary DOUBLE(7,2) ,
        commission DOUBLE(7,2) ,
                PRIMARY KEY (EMPNO));
```

Alternatively, the same query can also be re-written as,

```
SQL>CREATE TABLE Employee (
        empno  INT(4) PRIMARY KEY,
                ename  VARCHAR(10),
                salary  DOUBLE(7,2)
                commission DOUBLE(7,2) );
```

By alteration operation constraint can be added to the already defined database table.

```
SQL> ALTER TABLE DEPARTMENT
     ADD CONSTRAINT dept_pk
     PRIMARY KEY (deptno);
```

*** PRIMARY KEY is by default “NOT NULL”, means as per this given example “empno” cannot be null, hence every row MUST HAVE employee’s number/ ID. The NOT NULL constraint ensures that “NULL” values are not allowed to be under any column. NOT NULL is declared explicitly while defining the table.

```
SQL>CREATE TABLE Employee (
                empno  INT(4) PRIMARY KEY,
                ename  VARCHAR(10) NOT NULL,
                salary  DOUBLE(7,2)  NOT NULL,
                commission DOUBLE(7,2) );
```

The Employee table will return as,

Employee

EMPNO	ENAME	SALARY	COMM
7839	KING	7000.50	20.00
7698	BLAKE	2000.00	25.00
7782	CLARK	1900.00	

7566	**JONES**	**10000,00**	

"EMPNO" is declared as a PRIMARY KEY constraint, as a result this attribute becomes "NOT NULL" automatically for all the records in the table. However, among the other attributes, "ENAME" and "SALARY" is explicitly declared as NOT NULL, means for every record in the "Employee" table these two attributes can not consider any "NULL" or, missing value. On the other hand, attribute "COMM" is nullable means there can be employees who may/may not have any commission information to get stored in the database table.

Even though there can only be one PRIMARY KEY in a table, more than one UNIQUE KEY can be declared while defining the table to restrict the repetition of same data under any attribute.

*** PRIMARY KEY is a special type of UNIQUE KEY, but not all the UNIQUE KEYS can be considered as the PRIMARY KEY of the table.

```
SQL>CREATE TABLE   Department(
                   deptno INT(2) PRIMARY KEY,
                   dname VARCHAR(14) NOT NULL,
                   loc VARCHAR(13) NOT NULL,
                   CONSTRAINT dept_dname_uk UNIQUE(dname));
```

The "Department" table will return as,

Department

DEPTNO	DNAME	LOC
10	**ACCOUNTING**	**CLAPHAM**
30	**RESEARCH**	**NEWHAM**
40	**SALES**	**TOWER HAMLET**

Now, if another record that any database user wants to insert as,

50	**SALES**	**TOWER HAMLET**

Then this operation will not be executed because "SALES" department already exists in this table.

FOREIGN KEY constraint is used to define the relationships by REFERANCING between two/ more than two tables so that the data among those table can be accessed at the same time through a single query.

Let's assume, there is a "Department" table which is referenced in the "Employee" through FOREIGN KEY constraint.

The foreign key is defined in the child table, and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- FOREIGN KEY is used to define the column in the child table at the table constraint level.
- REFERENCE identify the table and column in the parent table.
- ONLY the "PRIMARY KEY CONSTRIANT" of a child table be able to be REFENCED as a FOREIGN KEY of any other table

However, in the example below the child table will be created. Let's assume "Department" table is the child table.

```
SQL> CREATE TABLE   Department(
                    deptno INT(2) PRIMARY KEY,
                    dname VARCHAR(14) NOT NULL,
                    loc VARCHAR(13) NOT NULL,
                    CONSTRAINT dept_dname_uk UNIQUE(dname));
```

Now, if we create "Employee" table and that is in a relationship with the table "Department" then we can define the parent table "Employee" as,

```
SQL> CREATE TABLE Employee(
                        empno  INT(4) PRIMARY KEY,
                        ename  VARCHAR(10) NOT NULL,
                        salary  DOUBLE(7,2)  NOT NULL,
                        commission DOUBLE(7,2),
                        deptno INT(2),
 FOREIGN KEY Employee (deptno) REFERENCES Department(deptno));
```

The constraint(s) specific database tables have "rules" for the data accessing and this is how the operations become more effective than usual.

DML Operations:

Data manipulation language (DML) is a kind of transaction on the database table which can:

- Add rows in the database table
- Changes/ update existing rows
- Remove existing rows of the table

***This is important to remember that, any DML operation impacts row or, set of rows and not the whole table or, the object.

To illustrate we can see the following example,

Salary

Salary_Grade	Low_Sal	High_Sal
1	1000	2500
2	3000	6000
3	6001	7000

Here, if there is a new salary grade that is required to be there in the existing salary table then it can be included simply by "Insertion" operation and that doesn't impact on the table structurally.

Let's consider the new grade is "5" and the low and high salary scale for this grade is 10000 and 12000 respectively.

SQL> INSERT INTO SALARY VALUES (5, 10000, 12000);

1 row inserted.

Salary

Salary_Grade	Low_Sal	High_Sal
1	1000	2500
2	3000	6000
3	6001	7000
5	10000	12000

The new row holds all the information about grade “5” salary.

However, in the existing table if the highest limit of the salary, “High_Sal” of grade “1” requires to get increased as 3000 then we have to perform “update” operation as shown below:

```
SQL> UPDATE SALARY
     SET High_Sal = 3000
     WHERE Salary_Grade=1;
1 row updated.
```

Salary

Salary_Grade	Low_Sal	High_Sal
1	1000	3000
2	3000	6000
3	6001	7000
5	10000	12000

If there is any requirement of removing any record from the table then “Deletion” operation is used. For example, if grade “3” salary is no longer required then that might get removed:

```
SQL>DELETE FROM SALARY
     WHERE GRADE=3;
1 row deleted.
```

Salary

Salary_Grade	Low_Sal	High_Sal
1	1000	3000
2	3000	6000
5	10000	12000

In order to access the data/records in the database table we use “SELECT” statement.

Now, from the above example if we want to retrieve the existing "salary grades" present in the salary table then the execution of the query will be,

SQL> SELECT Salary_Grade

FROM Salary;

Salary_Grade
1
2
5

All the salary grades are displayed.

Another example, if we want to retrieve all the details of the salary grade "2" only then we can restrict the other rows by adding "WHERE" clause while querying as follows:

SQL> SELECT *

FROM Salary

WHERE Salary_Grade=2;

Salary

Salary_Grade	Low_Sal	High_Sal
2	3000	6000