Merit-Quality-Excellence

IBA Institute of Emerging Technologies Khairpur

DATABASE MANAGEMENT SYSTEM

LAB No: 07

Instructor: Marina Gul

Objective of Lab No. 7:
After performing lab 7, students will be able to:

- To learn Data Definition Language (DDL)
- To learn an application of Constraints on a Table & its columns

Introduction to data definition Language

DDL is short name of Data Definition Language, defines the database structure or database schema. DDL also defines additional properties of the data defined in the database, as the domain of the attributes. The Data Definition Language also provide the facility to specify some constraints that would maintain the data consistency.
Following are the common SQL commands:

- CREATE - to create a database and its objects like (table, index, views, store procedure, function, and triggers)
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- RENAME - rename an object

DDL statements automatically commit the current transaction; they cannot be rolled back.

CREATE COMMAND:

This command is used to create database & its objects. There are two common CREATE statements available in SQL:
1. CREATE DATABASE
2. CREATE TABLE

1. CREATE DATABASE

A Database is defined as a structured set of data. So, in SQL the very first step to store the data in a well-structured manner is to create a database. The CREATE DATABASE statement is used to create a new database in SQL.
Syntax:

```
CREATE DATABASE database_name;
```

2. CREATE TABLE

The CREATE TABLE statement is used to create a table in SQL. We know that a table comprises of rows and columns. So while creating tables we have to provide all the information to SQL about the names of the columns, type of data to be stored in columns, size of the data etc. Let us now dive into details on how to use CREATE TABLE statement to create tables in SQL.

Syntax:

```
CREATE TABLE table_name
(
column1 data_type(size),
column2 data_type(size),
column3 data_type(size),
. . .
);
```

Example:

```
CREATE TABLE patient
(
ID INT(11),
name VARCHAR(30),
phone VARCHAR(15)
);
```

ALTER COMMAND:

`alter` command is used for altering the table structure, such as,

- to add a column to existing table
- to rename any existing column
- to change datatype of any column or to modify its size.
- to drop a column from the table.

1. `ALTER` Command: Add a new Column

Using `ALTER` command we can add a column to any existing table. Following is the syntax,

```
ALTER TABLE table_name ADD(
column_name datatype);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
address VARCHAR(200)
);
```

The above command will add a new column `address` to the table **student**, which will hold data of type `varchar` which is nothing but string, of length 200.

2. `ALTER` Command: Add multiple new Columns

Using `ALTER` command we can even add multiple new columns to any existing table. Following is the syntax,

```
ALTER TABLE table_name ADD(
column_name1 datatype1,
column-name2 datatype2,
column-name3 datatype3);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
father_name VARCHAR(60),
mother_name VARCHAR(60),
dob DATE);
```

The above command will add three new columns to the **student** table.

3. `ALTER` Command: Add Column with default value

`ALTER` command can add a new column to an existing table with a default value too. The default value is used when no value is inserted in the column. Following is the syntax,

```
ALTER TABLE table_name ADD(
column-name1 datatype1 DEFAULT some_value
);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
dob DATE DEFAULT '01-Jan-99'
);
```

The above command will add a new column with a preset default value to the table **student**.

4. `ALTER` Command: Modify an existing Column

`ALTER` command can also be used to modify data type of any existing column. Following is the syntax,

```
ALTER TABLE table_name modify(
column_name datatype
);
```

Here is an Example for this,

```
ALTER TABLE student MODIFY(
address varchar(300));
```

Remember we added a new column `address` in the beginning? The above command will modify the `address` column of the **student** table, to now hold upto 300 characters.

5. `ALTER` Command: Modify multiple Columns

Using `ALTER` command we can even modify multiple columns to any existing table. Following is the syntax,

```
ALTER TABLE table_name
modify column_name datatype,
modify column_name2 datatype;
```

Here is an Example for this,

```
ALTER TABLE student
modify father_name VARCHAR(70),
modify mother_name VARCHAR(70);
```

The above command will modify the `father_name & mother_name` column of the **student** table, to now hold upto 70 characters.

6. `ALTER` Command: Rename a Column

Using `ALTER` command you can rename an existing column. Following is the syntax,

```
ALTER TABLE table_name CHANGE COLUMN
old_column_name new_column_name datatype;
```

Here is an example for this,

```
ALTER TABLE student CHANGE COLUMN
address location VARCHAR(20);
```

The above command will rename `address` column to `location`.

7. `ALTER` Command: Drop a Column

`ALTER` command can also be used to drop or remove columns. Following is the syntax,

```
ALTER TABLE table_name DROP
column_name;
```

Here is an example for this,

```
ALTER TABLE student DROP
address;
```

The above command will drop the `address` column from the table **student**.

8. `ALTER` Command: Drop Multiple Columns

`ALTER` command can also be used to drop or remove multiple columns. Following is the syntax,

```
ALTER TABLE table_name
DROP column_name,
DROP column_name2;
```

Here is an example for this

```
ALTER TABLE student
DROP address,
DROP phone;
```

The above command will drop `address & phone` column from the table **student**.

DROP COMMAND:

Drop command is used to delete an existing SQL database or its objects.

1. `DROP DATABASE` Command: Delete a Database

`DROP DATABASE` command is used to drop or remove SQL database. Following is the syntax,

```
DROP DATABASE database_name;
```

Note: Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

2. `DROP TABLE` Command: Delete an object (i.e. Table, Column) from a Database

`DROP TABLE` command is used to drop or remove table from database. Following is the syntax,

```
DROP TABLE table_name;
```

`Drop` column is already covered in alter command section.

TRUNCATE COMMAND:

The `TRUNCATE TABLE` command deletes the data inside a table, but not the table itself.
Following is the syntax,

```
TRUNCATE TABLE table_name;
```

RENAME COMMAND:

The rename command is used to change the name of an existing database object (like Table,Column) to a new name.
Renaming a table does not make it to lose any data is contained within it.
Following is the syntax,

```
RENAME TABLE current_name TO new_name;
```

You can also use command to rename a table name:

```
ALTER TABLE current_name RENAME new_name;
```

Note: RENAME TABLE, unlike ALTER TABLE, can rename multiple tables within a single statement:

RENAME TABLE does not work for TEMPORARY tables. However, you can use ALTER TABLE to rename temporary tables.
RENAME TABLE works for views, except that views cannot be renamed into a different database.

Introduction to constraints

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.
Constraints can be divided into the following two types,

1. **Column level constraints:** Limits only column data.
2. **Table level constraints:** Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

- DEFAULT

NOT NULL constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.
One important point to note about this constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

```
CREATE TABLE Student(s_id int NOT NULL, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will not take NULL value.

UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values. A **UNIQUE** constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Using UNIQUE constraint when creating a Table (Table Level)

Here we have a simple CREATE query to create a table, which will have a column **s_id** with unique values.

```
CREATE TABLE Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will only have unique values and won't take NULL value.

Using UNIQUE constraint after Table is created (Column Level)

```
ALTER TABLE Student ADD UNIQUE(s_id);
```

The above query specifies that **s_id** field of **Student** table will only have unique value.

Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Using PRIMARY KEY constraint at Table Level

```
CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age
int);
```

The above command will creates a PRIMARY KEY on the `s_id`.

Using PRIMARY KEY constraint at Column Level

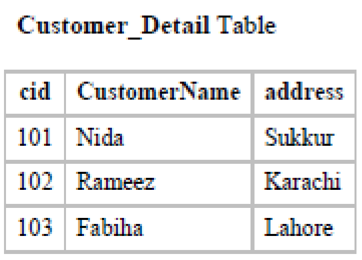
```
ALTER table Student ADD PRIMARY KEY (s_id);
```

The above command will creates a PRIMARY KEY on the `s_id`. For multiple columns following query:

```
PRIMARY KEY (s_id, home_city)
```

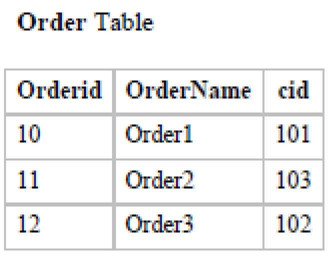
Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:



Customer_Detail Table

cid	CustomerName	address
101	Nida	Sukkur
102	Rameez	Karachi
103	Fabiha	Lahore



Order Table

Orderid	OrderName	cid
10	Order1	101
11	Order2	103
12	Order3	102

In **Customer_Detail** table, **cid** is the primary key which is set as foreign key in **Order** table. The value that is entered in **cid** which is set as foreign key in **Order** table must be present in **Customer_Detail** table where it is set as primary key. This prevents invalid data to be inserted into **cid** column of **Order** table.

If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

Using FOREIGN KEY constraint at Table Level

```
CREATE table Order(
Orderid int PRIMARY KEY,
OrderName varchar(60) NOT NULL,
cid int(11),
FOREIGN KEY(cid) REFERENCES Customer_Detail(cid)
);
```

In this query, **cid** in table Order is made as foriegn key, which is a reference of **cid** column in Customer_Detail table.

Using FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail ADD FOREIGN KEY (cid) REFERENCES
Customer_Detail(cid);
```

For multiple columns:

```
FOREIGN KEY(ord_no,book_id) REFERENCES neworder(ord_no,book_id),
```

Behavior of Foreign Key Column on Delete/Update

When two tables are connected with Foreign key, and certain data in the main table is deleted/updated, for which a record exists in the child table, then we must have some mechanism to save the integrity of data in the child table.

MySQL allows creating a table with CASCADE, SET NULL and RESTRICT options.

CASCADE option deletes or updates the row from the parent table (containing PRIMARY KEYs), and automatically delete or update the matching rows in the child table (containing FOREIGN KEYs).

RESTRICT option bars the removal (i.e. using delete) or modification (i..e using an update) of rows from the parent table.

SET NULL option will delete or update the row from the parent table, and set the foreign key column or columns in the child table to NULL.
You can use SET NULL for DELETE as well as UPDATE.
If SET NULL is used, you should not set NOT NULL options to the columns of the child table (containing FOREIGN KEYS).

```
CREATE table Order_Details(
Orderid int PRIMARY KEY,
OrderName varchar(60) NOT NULL,
id int(11),
FOREIGN KEY(id) REFERENCES customer(id)
ON UPDATE CASCADE ON DELETE RESTRICT
);
```

```
CREATE table Order_Details(
Orderid int PRIMARY KEY,
OrderName varchar(60) NOT NULL,
id int(11),
FOREIGN KEY(id) REFERENCES customer(id)
ON UPDATE CASCADE ON DELETE SET NULL
);
```

`CHECK` Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. It's like condition checking before saving data into a column.

Using `CHECK` constraint at Table Level

```
CREATE table Student(
s_id int NOT NULL CHECK(s_id > 0),
Name varchar(60) NOT NULL,
Age int
);
```

The above query will restrict the **s_id** value to be greater than zero.

Using `CHECK` constraint at Column Level

```
ALTER table Student ADD CHECK(s_id > 0);
```

CHECK CONSTRAINT using IN operator

MySQL CHECK CONSTRAINT can be applied to a column of a table, to set a limit for storing values within a range, along with IN operator.

```
CREATE TABLE customer(
id INT(11) NOT NULL PRIMARY KEY,
name VARCHAR(30),
phone INT(15),
country varchar(25) NOT NULL CHECK(country IN('UK', 'USA'))
);
```

You can use CHECK CONSTRAINT with LIKE, OR & AND operator and also with CASE statement.

`AUTO_INCREMENT` Constraint

MySQL allows you to set AUTO_INCREMENT to a column. Doing so will increase the value of that column by 1 automatically, each time a new record is added.

```
ALTER TABLE customer MODIFY id int AUTO_INCREMENT;
```

The above command will modify the id column of the customer table, to an auto increment.

`Default` Constraint

The DEFAULT constraint is used to set a default value for a column. The default value will be added to all new records, if no other value is specified.

SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
 ID int NOT NULL,
 LastName varchar(255) NOT NULL,
 FirstName varchar(255),
 Age int,
 City varchar(255) DEFAULT 'Sandnes'
);
```

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

Lab Task(s):

Exercise

Using Employee table, solve the following queries (1-5).

1. Create a replica of Employee table with all the records in it.

2. Add a column ‘Permanent Address’ in it.

3. Drop column ‘Address’ from it.

4. Add columns ‘House No’ character ,’Street No’ numeric, ’Area’ character ,’City’ character in it with the respective data types.

5. Change the data type of ‘House No’ from character to numeric.

6. Create the Data Definitions for each of the relations shown below, using SQL DDL. Assume the following attributes and data types:

FACULTY:

```
FacultyID (integer, primary key)
FacultyName (25 characters)
```

COURSE:

```
CourseID (8 characters, primary key)
CourseName (15 characters)
```

CLASS:

```
ClassID (8 characters)
CourseID (8 characters foreign key)
SectionNo (integer)
Semester (10 characters)
```

STUDENT:

```
StudentID (integer, primary key)
StudentName (25 characters)
FacultyID (integer foreign key)
```

7. How would you add an attribute, CLASS, to the STUDENT table?

8. Write a SQL statement to rename the table department to dept (with both methods).

9. Write a SQL statement to add a column regionId to the table locations.

10. Write a SQL statement to change the name of the column state_province to state in locations table, keeping the data type and size same.

11. Create a table DEPARTMENT with the following attributes:

    DEPTNO number, DNAME varchar(10), LOC varchar(10).

    PRIMARY KEY constraint on DEPTNO.

12. Create a table EMPLOYEE with the following attributes:

    EMPNO number, ENAME varchar(10),SAL number, DEPTNO number.

    Apply FOREIGN KEY constraint on DEPTNO referencing the DEPARTMENT table created in question 1 and PRIMARY KEY constraint on EMPNO and DEPTNO.

13. ALTER table EMPLOYEE created in question 2 and apply the constraint CHECK on ENAME attribute such that ENAME should always be inserted in capital letters.

14. ALTER table DEPARTMENT created in question 1 and apply constraint on DNAME such that DNAME should not be entered empty.

15. ALTER table EMPLOYEE created in question 2 and apply the constraint on SAL attribute such that no two salaries of the employees should be similar.

16. ALTER table EMPLOYEE created in question 2 and apply the constraint on DEPTNO attribute such that on update, update a child value and on delete set null value to a child.

END