Lab Manual for Database Management System

Lab No. 7
DDL and SQL Constraints

LAB 6: DDL AND SQL CONSTRAINTS

1. INTRODUCTION:

Data Definition Language (DDL) constitutes a fundamental aspect of database management, providing a set of commands used to define and manage the structure of a database. In essence, DDL serves as the architect's blueprint, allowing database administrators to articulate the framework for organizing and storing data efficiently. Through a concise yet powerful set of statements, DDL enables the creation, modification, and deletion of database objects, such as tables, indexes, and constraints. This control over the database structure empowers users to establish a logical and coherent data model, ensuring data integrity, consistency, and optimal performance. As an integral component of SQL (Structured Query Language), DDL plays a pivotal role in shaping the foundation upon which robust and well-organized databases are built.

2. OBJECTIVE:

After completing this lab the student should be able to:

- ***a.*** *Effectively employ diverse DDL SQL data types for creating optimized database tables.*
- ***b.*** *Demonstrate expertise in choosing and implementing appropriate DDL data types to ensure data accuracy and efficiency in storage.*
- ***c.*** *Use DDL to define well-structured tables and enhance data storage capabilities*

3. THEORY

Data Definition Language (DDL) in SQL serves as a crucial subset focused on defining and managing database structures and related metadata. Among the key commands, the fundamental operations involve creating and dropping database as well as creating, altering, and dropping tables. This brief overview will delve into the syntax and usage of of all the mentioned functions and their syntax used.

a. CREATE DATABASE:

The CREATE DATABASE command establishes a new database within the database server. The syntax is as follows:

Syntax → ***CREATE DATABASE <database_name>;***

Example:

```
CREATE DATABASE SalesDB;
```

This example creates a new database named "SalesDB."

b. DROP DATABASE:

Conversely, the DROP DATABASE command removes an existing database irreversibly. Exercise caution, as it doesn't request confirmation before deletion. The syntax is:

Syntax → ***DROP DATABASE <database_name>;***

Example:

```
DROP DATABASE SalesDB;
```

This example permanently deletes the "SalesDB" database. Ensure the necessity of deletion, as this action is not reversible.

c. CREATE TABLE:

The 'CREATE TABLE' statement is a fundamental DDL command used to create tables in a database. The syntax is as follows:

Syntax →
```
CREATE TABLE <table-name> (
<column1-definition>,
<column2-definition>,
…… …… …… …… …… …… …… …… …. );
```

Column definitions include the column name followed by its data type. For example:

```
CREATE TABLE PERSON (
  PERSONID INT,
  LNAME VARCHAR(20),
  FNAME VARCHAR(20) NOT NULL,
  DOB DATE,
  PRIMARY KEY(PERSONID));
```

This example creates a 'PERSON' table with columns for person ID, last name, first name, date of birth, and a primary key constraint on the 'PERSONID' column.

d. DROP TABLE:

The 'DROP TABLE' statement removes a table from the database permanently. The syntax is straightforward:

Syntax → ***DROP TABLE <table-name>;***

For instance:

```
DROP TABLE PERSON;
```

This command deletes the 'PERSON' table, and caution should be exercised as it is irreversible.

e. ALTER TABLE:

The 'ALTER TABLE' statement allows modifications to existing tables, useful when data is present, and recreating the table is impractical. Syntax varies across databases, and certain limitations may apply. Common alterations include adding, dropping, or modifying columns.

- ***Add a field:***

Syntax → ***ALTER TABLE <table-name> ADD <field-name> <data-type>;***

For instance:

```
ALTER TABLE PERSON ADD EMAIL VARCHAR(50);
```

This example adds an 'EMAIL' column of type VARCHAR(50) to the 'PERSON' table.

- ***Drop a field:***

Syntax → ***ALTER TABLE <table-name> DROP <field-name>;***

For instance:

```
ALTER TABLE PERSON DROP DOB;
```

Note: Some databases may restrict dropping fields, primarily serving for constraints removal.

- ***Modify a field:***

Syntax → ***ALTER TABLE <table-name> ALTER COLUMN <field-name> <new-field-declaration>;***

For instance:

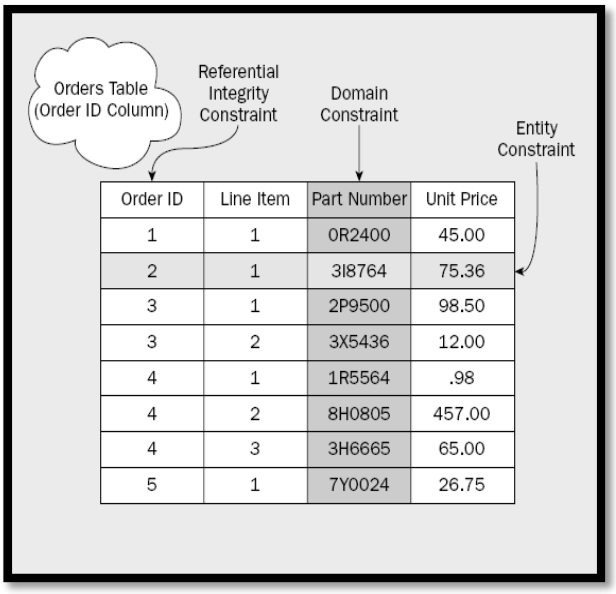
```
ALTER TABLE PERSON ALTER COLUMN FNAME VARCHAR(30);
```

This example modifies the 'FNAME' column, changing its data type to VARCHAR(30).

This is often limited and mainly used for adjusting constraints.

SQL Constraints

A constraint is a restriction. Placed at either column or table level, a constraint ensures that your data meets certain data integrity rules. There are a number of different ways to implement constraints, but each of them falls into one of three categories—entity, domain, or referential integrity constraints, as illustrated in Figure.



Order ID	Line Item	Part Number	Unit Price
1	1	0R2400	45.00
2	1	3I8764	75.36
3	1	2P9500	98.50
3	2	3X5436	12.00
4	1	1R5564	.98
4	2	8H0805	457.00
4	3	3H6665	65.00
5	1	7Y0024	26.75

- **DOMAIN CONSTRAINTS**

Domain constraints deal with one or more columns. What we're talking about here is ensuring that a particular column or set of columns meets particular criteria. When you insert or update a row, the constraint is applied without respect to any other row in the table—it's the column's data you're interested in. For example, if we want to confine the UnitPrice column only to values that are greater than or equal to zero, that would be a domain constraint. While any row that had a UnitPrice that didn't meet the constraint would be rejected, we're actually enforcing integrity to make sure that entire column (no matter how many rows) meets the constraint. The domain is the column, and our constraint is a domain constraint.

- **ENTITY CONSTRAINTS**

Entity constraints are all about individual rows. This form of constraint doesn't really care about a column as a whole; it's interested in a particular row, and would best be exemplified by a constraint that requires every row to have a unique value for a column or combination of problems.

- **REFERENTIAL INTEGRITY CONSTRAINTS**

Referential integrity constraints are created when a value in one column must match the value in another column—in either the same table or, far more typically, a different table.

- **PRIMARY KEY Constraints**

Primary keys are the unique identifiers for each row. They must contain unique values (and hence cannot be NULL). Because of their importance in relational databases, primary keys are the most fundamental of all keys and constraints.There are two ways. You can create the primary key either in your CREATE TABLE command or with an ALTER TABLE command.

Creating the Primary Key at Table Creation

The basic CREATE TABLE statements as discussed before are:

```
CREATE TABLE Customers (
        CustomerNo int IDENTITY NOT NULL,
        CustomerName varchar(30) NOT NULL,
        Address1 varchar(30) NOT NULL,
        Address2 varchar(30) NOT NULL,
        City varchar(20) NOT NULL,
        State char(2) NOT NULL,
        Zip varchar(10) NOT NULL,
        Contact varchar(25) NOT NULL,
        Phone char(15) NOT NULL,
        FedIDNo varchar(9) NOT NULL,
        DateInSystem smalldatetime NOT NULL)
```

To alter our CREATE TABLE statement to include a PRIMARY KEY constraint, we just add in the constraint information right after the column(s) that we want to be part of our primary key. In this case, we would use:

```
CREATE TABLE Customers (
        CustomerNo int IDENTITY NOT NULL PRIMARY KEY,
        CustomerName varchar(30) NOT NULL,
        Address1 varchar(30) NOT NULL,
        Address2 varchar(30) NOT NULL,
        City varchar(20) NOT NULL,
        State char(2) NOT NULL,
        Zip varchar(10) NOT NULL,
        Contact varchar(25) NOT NULL,
        Phone char(15) NOT NULL,
        FedIDNo varchar(9) NOT NULL,
        DateInSystem smalldatetime NOT NULL);
```

Creating a Primary Key on an Existing Table

Now, what if we already have a table and we want to set the primary key? For that we use PERSON table:

```
ALTER TABLE PERSON
ADD CONSTRAINT PK_ PERSONID PRIMARY KEY (PERSONID)
```

Our ALTER command tells SQL Server: That we are adding something to table (we could also be dropping something from table if we so chose)? What it is that we're adding (a constraint)? What we want to name the constraint (to allow us to address the constraint directly later)? The type of constraint (PRIMARY KEY)? The column(s) that the constraint applies to?

- **FOREIGN KEY Constraints**

Foreign keys are both a method of ensuring data integrity and a manifestation of the relationships between tables. When you add a foreign key to a table, you are creating a dependency between the table for which you define the foreign key (the referencing table) and the table your foreign key references (the referenced table). After adding a foreign key, any record you insert into the referencing table must either have a matching record in the referenced column(s) of the referenced table, or the value of the foreign key column(s) must be set to NULL.

Syntax →

```
<column name> <data type> <nullability>
FOREIGN KEY REFERENCES <table name>(<column name>)
```

Creating a Table with a Foreign Key

For the moment, we're going to ignore the ON clause. That leaves us, for our Orders table, with a script that looks something like this:

```
CREATE TABLE Orders (
        OrderID int IDENTITY NOT NULL PRIMARY KEY,
        CustomerNo int NOT NULL FOREIGN KEY REFERENCES Customers(CustomerNo),
       OrderDate smalldatetime NOT NULL,
        EmployeeID int NOT NULL);
```

Adding a Foreign Key to an Existing Table

```
ALTER TABLE Orders
ADD CONSTRAINT FK_EmployeeCreatesOrder
FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
```

- **UNIQUE Constraints**

UNIQUE constraints are essentially the younger sibling of primary keys in that they require a unique value throughout the named column (or combination of columns) in the table. You will often hear UNIQUE constraints referred to as alternate keys. The major differences are that they are not considered to be the unique identifier of a record in that table (even though you could effectively use it that way) and that you can have more than one UNIQUE constraint (remember that you can only haveone primary key per table).

Unlike a primary key, a UNIQUE constraint does not automatically prevent you from having a NULL value. Whether NULLs are allowed or not depends on how you set the NULL option for that column in the table. Keep in mind, however, that, if you do allow NULLs, you will be able to insert only one of them (although a NULL doesn't equal another NULL, they are still considered to be duplicate from the perspective of a UNIQUE constraint).

```
CREATE TABLE Shippers (
        ShipperID int IDENTITY NOT NULL  PRIMARY KEY,
        ShipperName varchar(30) NOT NULL,
        Address varchar(30) NOT NULL,
        City varchar(25) NOT NULL,
        State char(2) NOT NULL,
        Zip varchar(10) NOT NULL,
        PhoneNo varchar(14) NOT NULL UNIQUE);
```

- **CHECK Constraints**

The nice thing about CHECK constraints is that they are not restricted to a particular column. They can be related to a column, but they can also be essentially table- related in that they can check one column against another as long as all the columns are within a single table, and the values are for the same row being updated or inserted. They may also check that any combination of column values meets a criterion. The constraint is defined using the same rules that you would use in a WHERE clause. Examples of the criteria for a CHECK constraint include:

Goal	SQL
Limit `Month` column to appropriate numbers	`BETWEEN 1 AND 12`
Proper SSN formatting	`LIKE '[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9]'`
Limit to a specific list of `Shippers`	`IN ('UPS', 'Fed Ex', 'USPS')`
Price must be positive	`UnitPrice >= 0`
Referencing another column in the same row	`ShipDate >= OrderDate`

Let's create check constraint;

```
ALTER TABLE Customers
ADD CONSTRAINT CN_CustomerDateInSystem CHECK
(DateInSystem <= GETDATE ())
```

Now try to insert a record that violates the CHECK constraint; you'll get an error:

```
INSERT INTO Customers
(CustomerName, Address1, Address2, City, State, Zip, Contact, Phone, FedIDNo, DateInSystem)
VALUES
('Customer1', 'Address1', 'Add2', 'MyCity', 'NY', '55555',
'No Contact', '553-1212', '930984954', '12-31-2049')
```

4. ACTIVITY TIME BOXING

Activity Name	Activity Time	Total Time
Instruments Allocation + Setting up Lab	10 mints	10 mints
Walk through Theory & Tasks (Lecture)	60 mints	60 mints
Implementation & Practice time	90 mints	80 mints
Evaluation Time	20 mints	20 mints
	Total Duration	180 mints

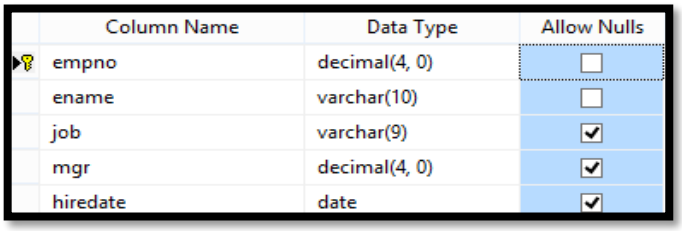
5. EXERCISE:

PERFORMANCE TASKS:

1. ***All Example Tasks.***

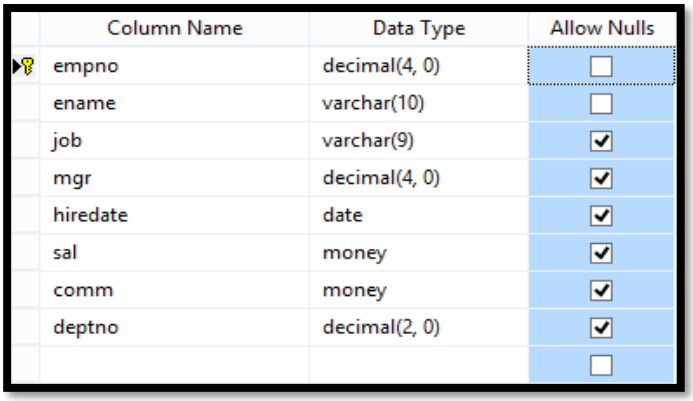
LAB FILE TASKS:

1. ***Complete the following tasks.***
   a. *Create Database named as Information.*
   b. *Using Database Information create Employee table based on the following design:*



Column Name	Data Type	Allow Nulls
empno	decimal(4, 0)	☐
ename	varchar(10)	☐
job	varchar(9)	☑
mgr	decimal(4, 0)	☑
hiredate	date	☑

   c. *Modify Employee table and add three more columns:*



Column Name	Data Type	Allow Nulls
empno	decimal(4, 0)	☐
ename	varchar(10)	☐
job	varchar(9)	☑
mgr	decimal(4, 0)	☑
hiredate	date	☑
sal	money	☑
comm	money	☑
deptno	decimal(2, 0)	☑
		☐

   d. *Insert 5 records in Employee Table.*
   e. *Delete all record from Employee table*
   f. *Drop Employee tabel Table.*
   g. *Drop Database Information.*

2. *Create a database for an online bookstore. In the database create the table will store information about books available for sale. Each book has a unique ISBN (International Standard Book Number), a title, an author, a genre, and a price.*
3. *Design a database for a small company to manage its employees. You need to create a table to store employee details, including their employee ID, name, department, position, and salary.*

4. *Design a CRM database schema with tables for customers, orders, and products. Include fields for CustomerID, Name, Email, and Phone in the customer table. Define fields like OrderID, CustomerID (referencing Customers), OrderDate, and TotalAmount in the orders table. In the product table, include ProductID, Name, Description, and Price. Establish relationships between tables to streamline customer interactions, order processing, and product sales management in the CRM system.*

5. *Create the following tables given in diagram with constraints.*