Learning Objectives:

- Understanding and Implementing Data Manipulation Language.
- Implementing SQL Basics-DML (Constraints, PK, FK, Select)

1. Introduction:

So far, we have successfully learned how to define a database and implemented various DDL statements on it but before starting today's learning, we should know the types of data and which languages are required for them.

Data	Description	Language
Structured	**Rigid format; Rows and Columns**	**SQL**
Semi-Structured	**Mix of consistent characteristics and data is not in a rigid structure. Can be hierarchal or metadata.**	**SQL**
Unstructured	**Not structured.**	**NoSQL**

In this course, we will be working on **structured** data. As mentioned above, structured data has a rigid form of rows and columns. And the database that is in the form of rows and columns in referred to as ***Relational Database.***

The database we create in SQL server is also a relational database, as you can expand it and see "Tables" option in it. Now, it is time that we enter data into the tables.

The data is entered in the table in the form of rows. Rows are also called ***Entries***, ***Records*** or ***Tuples***. The columns are called ***Fields*** or ***Attributes***. While the tables are ***Entities*** or ***Relations***.

To achieve the aim, we will learn about the *language for describing data and its relationship in a database, called* ***Data Manipulation Language (DML).***

1. Data Manipulation Language:

Some of the DML statements are:

- **INSERT** to insert a new row.
- **UPDATE** to update an existing row.
- **DELETE** to delete a row.

1- INSERT:

This query is used to add data into the tables.

Syntax:

```
INSERT into tableName VALUES (value1, value2, …)      -- Insert data into Table
```

If you want to specify both column name and the values to be inserted, use this form

```
INSERT into tableName (column1, column2, …)
VALUES (value1, value2, …)                    -- Inserts data into Table
```

Example:

We want to add values in our previously maintained table of Students, as follows

ID	FirstName	LastName	Age	Gender

We will use the following statement:

```
INSERT INTO Students (1, 'Ali', 'Akram', 24, 'Male')
```

After execution, the table will look like this:

ID	FirstName	LastName	Age	Gender
1	Ali	Akram	24	Male

Write the query for complete the table.

ID	FirstName	LastName	Age	Gender
1	Ali	Akram	24	Male
2	Ahmed	Khan	22	Male
3	Ayesha	Batool	21	Female

Insert Data Only in Specified Columns

It is also possible to only add data in specific columns.

The following SQL statement will add a new row, but only add data in the Id", "LastName" and the "FirstName" columns:

```
INSERT INTO Students (Id, LastName, FirstName) VALUES (4, 'Akhtar', 'Bisma')
```

Now the table will look like this:

ID	FirstName	LastName	Age	Gender
1	Ali	Akram	24	Male
2	Ahmed	Khan	22	Male
3	Ayesha	Batool	21	Female
4	Bisma	Akhtar		

2- UPDATE:

The UPDATE statement is used to update existing records in a table.

Syntax:

```
UPDATE table_name
SET column1=value, column2=value2,... WHERE some_column=some_value      -- Update value of a column
```

Notice the ***WHERE*** clause in the UPDATE statement. The ***WHERE*** clause specifies which record or records should be updated. If you omit the WHERE clause, all records will be updated!

Operators to be used with WHERE clause:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value, you want to return for at least one of the columns

Example:
We want to update the person "Ahmed, Khan" in the "Students" table. We use the following SQL statement:

UPDATE Students SET Age=23 WHERE LastName= 'Khan' AND FirstName='Ahmed'

3- DELETE

The DELETE statement is used to delete records/rows in a table.

Syntax:

```
DELETE FROM table_name WHERE some_column=some_value -- Update value of a column
```

Notice the ***WHERE*** clause in the DELETE statement. The ***WHERE*** clause specifies which record or records that should be deleted. If you omit the ***WHERE*** clause, all records will be deleted!

Example:
We want to delete the person "Bisma Akhtar" from the "Students" table. We use the following SQL statement:

DELETE FROM Students WHERE LastName= 'Akhtar' AND FirstName='Bisma'

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
```

Or

```
DELETE * FROM table_name
```

Or you can use **TRUNCATE** *statement.*

```
TRUNCATE Table table_name
```

Note: Be very careful when deleting records. You cannot undo this statement!

You are now able to compare **the Database Management System** *with* **File Handling Systems**, *and how the management of data is easy in DBMS instead of FHS where you have to write complex codes for insertion, deletion, and retrieval of data.*

2. DML Constraints:

SQL constraints are a set of rules implemented on tables in relational databases to dictate what data can be inserted, updated, or deleted in its tables. This is done to ensure the accuracy and the reliability of information stored in the table.

The following constraints are commonly used in SQL:

- **CHECK** - Ensures that the value in a column satisfies a specific condition.
- **NOT NULL** - Ensures that a column cannot have a NULL value.
- **UNIQUE** - Ensures that all values in a column are different.
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
- **FOREIGN KEY** - Prevents actions that would destroy links between tables.
- **DEFAULT** - Sets a default value for a column if no value is specified.

1- CHECK Constraint:

A check constraint in SQL Server (Transact-SQL) allows you to specify a condition on each row in a table.

Syntax:

```
CREATE TABLE table_name
(
column1 datatype [ NULL | NOT NULL ],
column2 datatype [ NULL | NOT NULL ]
…
CHECK [ NOT FOR REPLICATION] (column_name condition)
);
```

```
CREATE TABLE Teacher (Teacher_id INT NOT NULL, last_name VARCHAR(50), first_name VARCHAR(50) NOT NULL, salary MONEY, CONSTRAINT check_Teacher_id CHECK (Teacher_id BETWEEN 10000 and 50000) );
```

Now if we add a record of a teacher, his salary will be checked at runtime, and it won't allow to enter amount less than 10,000 or greater than 50,000.

2- Unique Constraint:

This constraint allows you to uniquely identify each row in the table. Which means, all tuples in a particular field will have unique/distinct values.

Syntax:

```
CREATE TABLE table_name
(
column1 datatype UNIQUE,
column2 datatype,
…
);
```

Example:

In Student table, we want to make ID a unique column. We use the following SQL statement:

```
CREATE TABLE  Student (ID INT UNIQUE, FirstName VARCHAR(10), LastName VARCHAR (20)…);
```

3- Primary Key:

Primary key is the constraint that uniquely identify each tuple in the table. If a field in a table is primary key, then it cannot have NULL values.

Rules for Primary Key:

- Each table can have **only one Primary Key.**
- All the **values are unique** and Primary key value can uniquely identify each row.
- The system will **not allow inserting** a row with a primary key **which already exists** in the table.
- Primary Key **cannot be NULL**.

Syntax:

```
CREATE TABLE table_name
(
column1 datatype PRIMARY KEY,
column2 datatype,
…
);
```

Example:

We want to create a course table, with a PRIMARY Key Course_Code. That can't be NULL and should be unique for every course. We use the following SQL statement:

```
CREATE TABLE  Course (Course_Code INT PRIMARY KEY, Course_Name VARCHAR(10), CreditHours INT );
```

Making an existing column a Primary key.

SQL will not allow you to make an attribute PRIMARY KEY, unless it is declared ***NOT NULL***.
So, first use ***ALTER*** statement to change the datatype of the attribute.

```
ALTER table  Student  ADD PRIMARY KEY(ID)
```

Inserting values in Table with Primary key:

Example: Let's see if it allows entering Multiple Records with Same Course ID.

Insert 4 rows with **different Course_Code**

```
Insert into COURSE values (1,'DBMS', 3)
Insert into COURSE values (2,'CN',3)
Insert into COURSE values (3,'DBMS',1)
Insert into COURSE values (4,'OS',2)
```

Try inserting new records with an **existing Course_code**

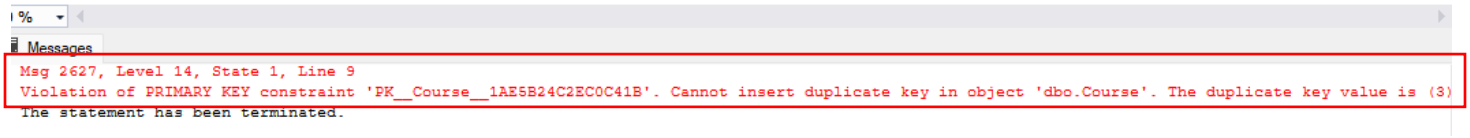
```
Insert into COURSE values (3,'SDA', 3)
```

Result:
Error. System does not allow inserting new value as 3 is already there in Course_ID column which is a Primary Key.

```
use SE_21

--CREATE TABLE Course (Course_Code INT PRIMARY KEY, Course_Name VARCHAR(10), CreditHours INT );
--Insert into COURSE values (1,'DBMS', 3)
--Insert into COURSE values (2,'CN',3)
--Insert into COURSE values (3,'DBMS',1)
--Insert into COURSE values (4,'OS',2)

Insert into COURSE values (3,'SDA', 3)
```



Interesting Facts!

- The Primary key can be a combination of multiple columns. This combination is known as the **Composite primary key**.
- The Primary key can have a maximum of 16 columns.

```
ALTER table TableName ADD PRIMARY KEY(column1, column2 )
```

4- Foreign Key:

Foreign key maintains the relationship between two or more tables. If two tables have any connection between them, the primary key of one table is put (referred) into other table. This Primary key, in the new table, is now called Foreign Key.

Rules for Foreign key

- **NULL is allowed** in Foreign key.
- The table being referenced is called the **Parent Table**
- The table with the foreign key is called **Child Table.**
- The foreign key in child table **references** the primary key in the parent table.
- This parent-child relationship enforces the rule which is known as "**Referential Integrity**"

Referential Integrity refers that a foreign key must either be Primary Key of some other table, or it must be null.

Making a foreign key in SQL server:

Syntax:

```
CREATE TABLE table_name
(
column1 datatype references some_tablename,
column2 datatype,
…
);
```

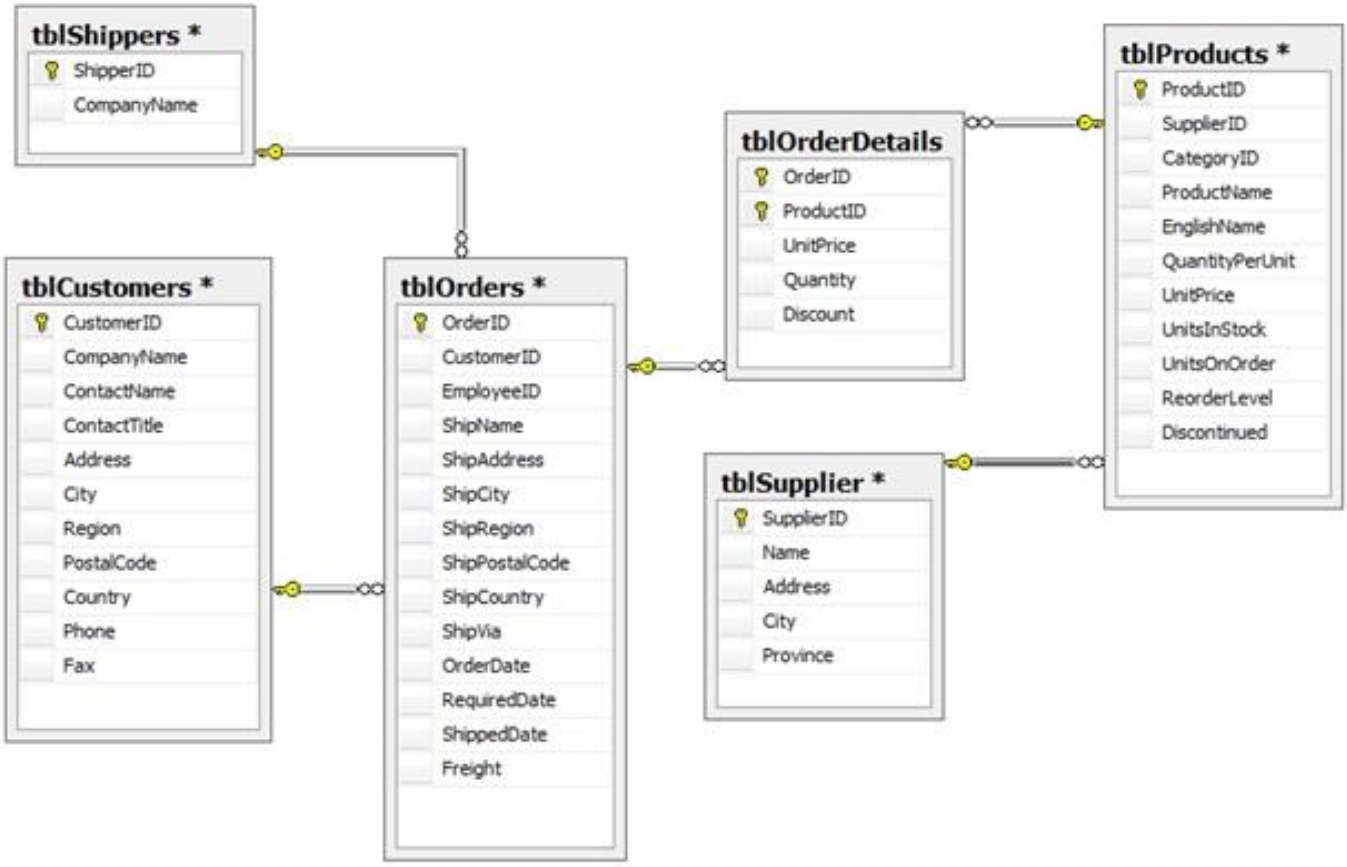
It is preferred to keep the name of PK and FK same. But the point to be noted is that the reference is not being created because of the name, instead it is being made by the *REFERENCES* keyword.

Example: Let's create a semester table having information of all the courses being taught in the

```
Create table teacher (semester_name int primary key, Session varchar(50), Course_Code int references Course)
```

3. Lab Tasks

1. *Create a Sales database with following tables. Notice that the arrows are the relationships between tables.*
2. *Create Primary key of each table, and foreign key of the referencing table.*
3. *Create at least 3 rows of data in each table.*
4. *Limit the amount of **Quantity per unit** between 1 to 10 only.*
5. *When you are done inserting the data delete the row for the order being shipped to UK having OrderID one.*
6. *Using a single command, write the SQL code that will enter the Discount =20 for all orders having unit price greater than 1000.*



4. Home Assignment:

- Insert at least 5 rows of data in each table of your management system.
- Identify and create Primary keys for each table.

Remember: If a table can't be uniquely identified by one column you can assign more than one PKs.

- Create Foreign keys in the tables which have a relationship with each other.