Learning outcome 3: Implement Database

3.1 Description to SQL

- **Introduction of SQL**

SQL stands for Structured Query Language**.**
SQL is a database computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System and all Relational Database Management Systems (RDBMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

SQL is widely popular because it can execute queries against a database, retrieve data , insert records, update records, delete records from a database, create new databases, create new tables in a database, create stored procedures in a database, create views in a database and set permissions on tables, procedures, and views.

- **SQL sub-languages**

SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP, etc.

SQL commands are grouped into below major categories (sublanguages) depending on their functionality:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Transaction Control Language (TCL)
4. Data Control Language (DCL)
5. Data Query Language (DQL)

- **Description of SQL commands per sublanguage**

1. Description of Data Definition Language commands:

Data definition language (DDL) refers to the set of SQL commands that can create and manipulate the structures of a database objects and database itself.

No	Command &description
1	**CREATE:** Creates a new table, a view of a table, or other object in the database.
2	**ALTER:** Modifies an existing database object, such as a table.
3	**DROP:** Deletes an entire table, a view of a table or other objects in the database
4	**RENAME:** Renames a database or a table by giving another name.
5	**TRUNCATE:** Is used to delete complete data from an existing table.

2. Data Manipulation Language commands:

DML commands work on records in a table. These are basic operations we perform on data such as inserting new records, deleting unnecessary records, and updating existing records.

Below are listed command under DML

No	Command & description
1	**INSERT**: Creates a record.
2	**UPDATE**: Modifies records.
3	**DELETE**: Deletes records.

3. Data Control Language commands

These are used by the database administrator to grant or revoke privileges to users of the RDBMS.

No	Command &description
1	**GRANT:** Gives a privilege to user
2	**REVOKE:** Takes back privileges granted from user.

4. Transaction Control Language (TCL):

The commands of SQL that are used to control the transactions made against the database. Below are listed command under TCL.

No	Command & description
1	**COMMIT**: used to save changes invoked by a transaction to the database
2	**ROLLBACK**: is used to undo the transactions that have not been saved in database.
3	**SAVEPOINT**: is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

C.5. Data Query Language (DQL):

This command is used to select the data from database tables; view, synonyms and sequence etc.

Below are listed command under DQL

Command	description
SELECT	Is used to retrieve records from one or more tables in your SQL database.

NOTE: TCL Commands (COMMIT, ROLLBACK, SAVEPOINT) are used for only **DML** Commands (INSERT, UPDATE, DELETE) while **DDL**, **DCL** commands are Auto-committed.

➢ SQL Operators

Introduction

An operator is a reserved word or a character that is used to query our database in a SQL expression. To query a database using operators, we use a WHERE clause.

Operators are necessary to define a condition in SQL, as they act as a connector between two or more conditions. The operator manipulates the data and gives the result based on the operator's functionality.

Types of Operator in SQL

SQL supports following types of operators: Ari*thmetic Operators, Assignment operators, Compound Operators, Logical Operators,* and *Bitwise Operators.*

- **Arithmetic operations**

These operators are used to perform operations such as addition, multiplication, subtraction etc.

Operator	Operation	Description
+	Addition	Add values on either side of the operator
–	Subtraction	Used to subtract the right hand side value from the left hand side value
*	Multiplication	Multiples the values present on each side of the operator
/	Division	Divides the left hand side value by the right hand side value
%	Modulus	Divides the left hand side value by the right hand side value; and returns the remainder

Examples:

```
mysql> SELECT 32+25 AS ADDITION;
+----------+
| ADDITION |
+----------+
|       57 |
+----------+
```

```
mysql> SELECT 24/12 AS DIVISION;
+----------+
| DIVISION |
+----------+
|   2.0000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT 32%3 AS MODULUS;
+---------+
| MODULUS |
+---------+
|       2 |
+---------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 32-25 AS SUBTRACTION;
+--------------+
| SUBTRACTION |
+--------------+
|            7 |
+--------------+
1 row in set (0.00 sec)

mysql> SELECT 32*2 AS MULTIPLICATION;
+-----------------+
| MULTIPLICATION |
+-----------------+
|              64 |
+-----------------+
```

- **Comparison operators**

A comparison (or relational) operator is a mathematical symbol which is used to compare two values.

Comparison operators are used in conditions that compares one expression with another. The result of a comparison can be TRUE, FALSE, or UNKNOWN (an operator that has one or two NULL expressions returns UNKNOWN).

Operator	Operation	Description
=	Equal to	Used to check if the values of both operands are equal or not. If they are equal, then it returns TRUE.
>	Greater than	Returns TRUE if the value of left operand is greater than the right operand.
<	Less than	Checks whether the value of left operand is less than the right operand, if yes returns TRUE.
>=	Greater than or equal to	Used to check if the left operand is greater than or equal to the right operand, and returns TRUE, if the condition is true.
<=	Less than or equal to	Returns TRUE if the left operand is less than or equal to the right operand.
<> or! =	Not equal to	Used to check if values of operands are equal or not. If they are not equal then, it returns TRUE.

Example: For your better understanding, I will consider the following table CUSTOMER to perform various operations.

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000
2	ISHIMWE	NAOME	25	KAMONYI	1500
3	ISHIMWE	SAMUEL	23	MUHANGA	2000
4	GATETE	YOUSSOUF	25	RUHANGO	6500
5	NISHIMWE	ALICE	27	NYANZA	4500
6	TUYIZERE	JOSIANE	22	HUYE	4500
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000

Example1. [Use equal to] (=):

SELECT * FROM CUSTOMER WHERE CSALARY=2000;

OUTPUT:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000
3	ISHIMWE	SAMUEL	23	MUHANGA	2000

Example2. [Not Equal to] (<>):

SELECT * FROM CUSTOMER WHERE CSALARY < >2000;

OUTPUT:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
2	ISHIMWE	NAOME	25	KAMONYI	1500
4	GATETE	YOUSSOUF	25	RUHANGO	6500
5	NISHIMWE	ALICE	27	NYANZA	4500
6	TUYIZERE	JOSIANE	22	HUYE	4500
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000

- **Assignment operators**

The assignment operator (=) in SQL Server is used to assign the values to a variable. The equal sign (=) is the only Transact-SQL assignment operator.

In the following example, we create @MyCounter variable and then the assignment operator sets @MyCounter variable to a value i.e. 1.

```
DECLARE @MyCounter INT;
SET @MyCounter = 1;
```

- **Logical operators**

The logical operators are used to perform operations such as ALL, ANY, NOT, BETWEEN etc.
Logical operators separate two or more conditions in the WHERE clause of an SQL statement.

Here is a list of all the logical operators available in SQL.

Operator	Description
ALL	Used to compare a specific value to all other values in a set
ANY	Compares a specific value to any of the values present in a set.
IN	Used to compare a specific value to the literal values mentioned.
BETWEEN	Searches for values within the range mentioned.
AND	Allows the user to mention multiple conditions in a WHERE clause.
OR	Combines multiple conditions in a WHERE clause.
NOT	A negate operators, used to reverse the output of the logical operator.
EXISTS	Used to search for the row's presence in the table.
LIKE	Compares a pattern using wildcard operators.
SOME	Similar to the ANY operator, and is used compares a specific value to some of the values present in a set.
UNIQUE	Searches every row of a specified table for uniqueness(no duplicate)

The different logical operators are shown below:

1. ALL OPERATOR

It is used to compare a value with every value in a list or returned by a query.
Must be preceded by **=, !=, >, < ,<=,** or **>=** evaluates.
ALL operators in the MySQL query are used to extract all tuples or records of the select statement.
ALL keyword is also used to make a comparison of a value with each and every data in another set of output from a subquery.

- The ALL operator outputs true if and only if the complete subqueries will satisfy the condition. ALL operator is headed by a comparison operator and will output true if all of the values of the subquery will fulfill the condition.
- ALL is always used in combination with select, where, having clause of the MySQL.
- ALL is used to select all records or rows of a select query. It compares the value of every value in a list or results from a query.

  For example, ALL means either greater than every value, means greater than the maximum value, less than every value or equal to ever value. Suppose ALL >(1, 2, 3) means greater than 3, ALL < (1,2,3) means less than 1.

Consider the CUSTOMER table as shown below:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000
2	ISHIMWE	NAOME	25	KAMONYI	1500
3	ISHIMWE	SAMUEL	23	MUHANGA	2000
4	GATETE	YOUSSOUF	25	RUHANGO	6500
5	NISHIMWE	ALICE	27	NYANZA	4500
6	TUYIZERE	JOSIANE	22	HUYE	4500
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000

Syntax of ALL operators

```
select All column_name1, column_name2 from table name where condition
```

mysql> SELECT ALL CID, CLAST_NAME, CADDRESS FROM CUSTOMER;

CID	CLAST_NAME	CADDRESS
1	KAZE	KIGALI
2	ISHIMWE	KAMONYI
3	ISHIMWE	MUHANGA
4	GATETE	RUHANGO
5	NISHIMWE	NYANZA
6	TUYIZERE	HUYE
7	UWIRAGIYE	NYAMAGABE

Below is the syntax of ALL operators with having or where clause:

```
select column_name1,column_name2 from table_name comparison operator
ALL(select column_name from table_name where condition);
```

EXAMPLE 1:

mysql> select CID, CLAST_NAME, CSALARY FROM CUSTOMER where CSALARY<
ALL(select CSALARY FROM CUSTOMER WHERE CSALARY>2000);

THE OUTPUT WILL BE:

CID	CLAST_NAME	CSALARY
1	KAZE	2000
2	ISHIMWE	1500
3	ISHIMWE	2000

EXAMPLE 2:

select CID, CLAST_NAME, CSALARY FROM CUSTOMER where CSALARY<=ALL(select CSALARY FROM CUSTOMER WHERE CSALARY>2000);

THE OUTPUT WILL BE:

CID	CLAST_NAME	CSALARY
1	KAZE	2000
2	ISHIMWE	1500
3	ISHIMWE	2000
5	NISHIMWE	4500
6	TUYIZERE	4500

2. ANY OPERATOR

EXAMPLE 1:

SELECT * FROM CUSTOMER WHERE CAGE>ANY (SELECT CAGE FROM CUSTOMER WHERE CAGE> 22);

THE OUTPUT WILL BE:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000
2	ISHIMWE	NAOME	25	KAMONYI	1500
4	GATETE	YOUSSOUF	25	RUHANGO	6500
5	NISHIMWE	ALICE	27	NYANZA	4500
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000

EXAMPLE 2:

Consider two tables customer and orders:

CUSTOMER:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000
2	ISHIMWE	NAOME	25	KAMONYI	1500
3	ISHIMWE	SAMUEL	23	MUHANGA	2000
4	GATETE	YOUSSOUF	25	RUHANGO	6500
5	NISHIMWE	ALICE	27	NYANZA	4500
6	TUYIZERE	JOSIANE	22	HUYE	4500
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000

ORDERS:

order_id	CID	order_date
2	1	2016-04-18
3	2	2016-04-19
4	2	2016-04-20
5	3	2016-05-01

Syntax:

SELECT num_value **FROM** table1 **WHERE** experession ANY (**SELECT** num_val **FROM** table2);

```
mysql>SELECT * FROM  CUSTOMER WHERE CID> ANY(SELECT CID FROM ORDERS WHERE CID>2);
```

THE OUTPUT WILL BE:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
4	GATETE	YOUSSOUF	25	RUHANGO	6500
5	NISHIMWE	ALICE	27	NYANZA	4500
6	TUYIZERE	JOSIANE	22	HUYE	4500
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000

3. IN, NOT IN

These operators in SQL are used with SELECT, UPDATE and DELETE statements/queries to select, update and delete only particular records in a table those meet the condition given in WHERE clause and conditions given in IN, NOT IN operators.
I.e. it filters records from a table as per the condition. Syntax for SQL IN & NOT IN operators are given below.

SYNTAX FOR AND, OR OPERATORS IN SQL:

SELECT *column_name(s)*FROM *table_name* WHERE **Where [condition]** IN (*value1*, *value2*, ...);

OR:

SELECT *column_name(s)* FROM *table_name* WHERE **Where [condition]** IN (*SELECT STATEMENT*);

Where [condition] should be in the following format:
[column_name] [Operator] [Value];

Where,
column_name: Any one of the column names in the table.
Operator: Any one of the following (>, <, =, >=, <=, NOT, LIKE etc)
Value: User defined value.

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

Examples:
(1) SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');
(2) select * from customer where cage IN (18,22,25);

OUTPUT WILL BE:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
2	ISHIMWE	NAOME	25	KAMONYI	1500
4	GATETE	YOUSSOUF	25	RUHANGO	6500
6	TUYIZERE	JOSIANE	22	HUYE	4500

4. The SQL BETWEEN, NOT BETWEEN Operators

The **BETWEEN** operator selects values within a given range. The values can be numbers, text, or dates.

The **BETWEEN** operator is inclusive: begin and end values are included.

- **BETWEEN Syntax:**

SELECT *column_name(s)*
FROM *table_name*WHERE *column_name* BETWEEN *value1* AND *value2;*

Examples:

The following SQL statement selects all products with a price between 10 and 20:
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;

- **NOT BETWEEN Syntax:**

SELECT *column_name(s)* FROM *table_name*WHERE *column_name NOT* BETWEEN *value1* AND *value2;*
Please consider the following table with few records as given below.
Table name (for example): student
Column names in this table: Student_ID, Student_name, City and Age

Available records: 4 rows

Student_ID	Student_name	City	Age
1	Raju	Chennai	21
2	Mani	Delhi	19
3	Thiyagarajan	Vizag	27
4	Surendren	Vizag	25

EXAMPLE1: HOW TO USE BETWEEN…AND IN A WHERE CLAUSE IN SELECT QUERIES

SQL query:

SELECT Student_ID, Student_name, City, Age from student
WHERE Age BETWEEN 19 AND 25;

SQL query Output:

Student_ID	Student_name	City	Age
1	Raju	Chennai	21
2	Mani	Delhi	19
4	Surendren	Vizag	25

EXAMPLE2:

HOW TO USE NOT BETWEEN…AND IN WHERE CLAUSE IN SELECT QUERIES

SQL query:

SELECT Student_ID, Student_name, City, Age from student WHERE Age NOT BETWEEN 19 AND 25;

SQL query Output:

Student_ID	Student_name	City	Age
3	Thiyagarajan	Vizag	27

5. AND, OR, NOT OPERATORS

The **WHERE** clause can be combined with **AND**, **OR**, and **NOT** operators.

The **AND** and **OR** operators are used to filter records based on more than one condition:

- The **AND** operator displays a record if all the conditions separated by **AND** are TRUE.
- The **OR** operator displays a record if any of the conditions separated by **OR** is TRUE.
- The **NOT** operator displays a record if the condition(s) is NOT TRUE.

❖ **AND Syntax**

SELECT *column1*, *column2, ...*FROM *table_name* WHERE *condition1* AND *condition2* AND *condition3 ...*;

- **OR Syntax**

SELECT *column1*, *column2, ...*FROM *table_name* WHERE *condition1* OR *condition2* OR *condition3 ...*;

- **NOT Syntax**

SELECT *column1*, *column2, ...*FROM *table_name* WHERE NOT *condition*;

EXAMPLES:

(a) AND Example

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

```
SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin';
```

(b) OR Example

The following SQL statement selects all fields from "Customers" where country is "Germany" OR "Spain":

```
SELECT * FROM Customers WHERE Country='Germany' OR Country='Spain';
```

(c) NOT Example

The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

```
SELECT * FROM Customers WHERE NOT Country='Germany'
```

Note: You can also combine the AND, OR and NOT operators.

Examples:

1. The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

2. The following SQL statement selects all fields from "Customers" where country is NOT "**Germany**" and NOT "**USA**":

```
SELECT * FROM Customers WHERE NOT Country='Germany' AND NOT Country='USA';
```

6. LIKE operator

The SQL LIKE operator is used to find matches between a character string and a specified pattern.

This operator is most commonly used in conjunction with two other SQL operators, WHERE and SELECT, to search for a value in a column.

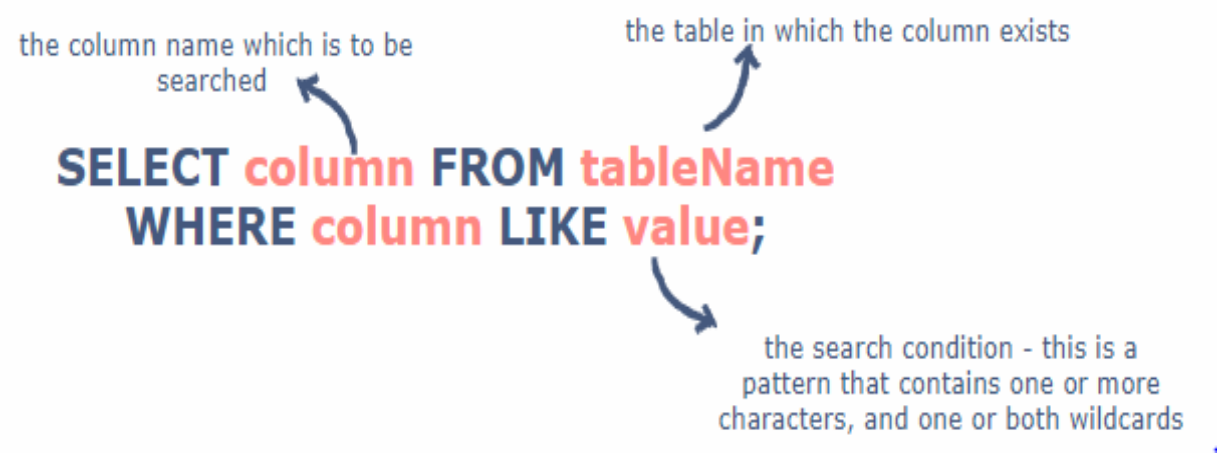
A **wildcard** is the pattern that specifies the search criteria.
The SQL LIKE operator uses two wildcards, either independently or in conjunction.

These wildcards are:

- **Percentage symbol (%)**, represents zero, one, or multiple characters.
- **Underscore symbol (_)**, represents a single character.

Note: MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).

Syntax:



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position

WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Here are some examples showing different **LIKE** operators with '%' and '_' wildcards:

SQL LIKE Examples:

❖ The following SQL statement selects all customers with a CustomerName starting with "a":
*SELECT * FROM Customers WHERE CustomerName LIKE 'a%';*

❖ The following SQL statement selects all customers with a CustomerName ending with "a":
*SELECT * FROM Customers WHERE CustomerName LIKE '%a';*

❖ The following SQL statement selects all customers with a CustomerName that have "or" in any position:
*SELECT * FROM Customers WHERE CustomerName LIKE '%or%';*

❖ The following SQL statement selects all customers with a CustomerName that have "r" in the second position:
*SELECT * FROM Customers WHERE CustomerName LIKE '_r%';*

❖ The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

*SELECT * FROM Customers WHERE CustomerName LIKE 'a_ _%';*

❖ The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":
*SELECT * FROM Customers WHERE ContactName LIKE 'a%o';*

❖ The following SQL statement selects all customers with a CustomerName that does NOT start with "a":
*SELECT * FROM Customers WHERE CustomerName NOT LIKE 'a%';*

7. The SQL EXISTS Operator

The **EXISTS** operator is used to test for the existence of any record in a subquery.
The **EXISTS** operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s) FROM table_name WHERE EXISTS (SELECT column_name
FROM table_name WHERE condition);
```

Example1:

```
SELECT cid,cage from customer where exists (select csalary from customer
where csalary=2000);
```

Output:

```
+------+------+
| cid  | cage |
+------+------+
|    1 |   32 |
|    2 |   25 |
|    3 |   23 |
|    4 |   25 |
|    5 |   27 |
|    6 |   22 |
|    7 |   24 |
+------+------+
```

Example2:

```
SELECT cid,cage from customer where exists (select csalary from customer where
csalary>200000);
```

Output:

```
Empty set (0.00 sec)
```

8. Some operator

It evaluates the condition between the outer and inner tables and evaluates to true if the final result returns **any one** row. If not, then it evaluates to false.

- The SOME and ANY comparison conditions are similar to each other and are completely interchangeable.
- SOME must match at least one row in the subquery and must be preceded by comparison operators.

Syntax:

SELECT column_name(s) FROM table_name WHERE expression comparison_operator SOME (subquery);

EXAMPLE:

SELECT * FROM CUSTOMER WHERE CAGE>SOME(SELECT CAGE FROM CUSTOMER WHERE CAGE> 22);

Output:

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000
2	ISHIMWE	NAOME	25	KAMONYI	1500
4	GATETE	YOUSSOUF	25	RUHANGO	6500
5	NISHIMWE	ALICE	27	NYANZA	4500
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000

9. UNIQUE operator

The **SQL UNIQUE** Operator is used to check whether the sub-query has any duplicate values in the result. It returns true if the sub-query has no duplicate values, else returns false.

Syntax:

The syntax for using UNIQUE operator is given below:

```
SELECT table1.colum FROM table1 WHERE UNIQUE (SELECT table2.column FROM table2
WHERE table1.column = table2.column);
```

Example:

Consider a database containing tables called ***Employee*** and ***Contact_Info*** with the following records:

Table 1: *Employee* table

EmpID	Name	City	Age	Salary
1	John	London	25	3000
2	Marry	New York	24	2750
3	Jo	Paris	27	2800
4	Kim	Amsterdam	30	3100
5	Ramesh	New Delhi	28	3000
6	Huang	Beijing	28	2800

Table 2: *Contact_Info* table

Phone_Number	EmpID	Address	Gender
+1-80XXXXX000	2	XXX, Brooklyn, New York, USA	F
+33-14XXXXX01	3	XXX, Grenelle, Paris, France	M
+31-20XXXXX19	4	XXX, Geuzenveld, Amsterdam, Netherlands	F
+86-10XXXXX458	6	XXX, Yizhuangzhen, Beijing, China	M
+65-67XXXXX4	7	XXX, Yishun, Singapore	M

+81-35XXXXX72	8	XXX, Koto City, Tokyo, Japan	M

To find all the employees whose contact information is updated in the *Contact_Info* table, the below mentioned SQL code can be used.

```
SELECT Employee.EmpID, Employee.Name FROM Employee WHERE UNIQUE (SELECT Contact_Info.EmpID FROM Contact_Info WHERE Employee.EmpID = Contact_Info.EmpID);
```

This will produce the following result:

EmpID	Name
2	Marry
3	Jo
4	Kim
6	Huang

- **Membership operators**

Membership operators are **operators used to validate the membership of a value**.
It tests for membership in a sequence, such as strings, lists, or tuples.
Set-membership tests: IN, NOT IN.

SELECT column1, column2....columnN FROM table_name WHERE column_name IN (val-1, val-2,...val-N); SQL NOT IN Clause Page 20 of 146 SELECT column1, column2....columnN FROM table_name WHERE column_name NOT IN (val-1, val-2,...val-N); EXAMPLE: / try it by yourself to see the result SELECT*FROM CUSTOMER WHERE CAGE NOT IN ('32','22');

- **Identity operators**

An identity column of a table is **a column whose value increases automatically**.
The value in an identity column is created by the server. A user generally cannot insert a value into an identity column.

Syntax:

IDENTITY [(seed, increm ent)]

Arguments:

1. **Seed:** Starting value of a column. The default value is 1.
2. **Increment:** It specifies the incremental value that is added to the identity column value of the previous row. The default value is 1.

The following shows an Identity property when the table is created:

```
Create Table Person ( PersonId int identity (1, 1), Name nvarchar (20) );
```

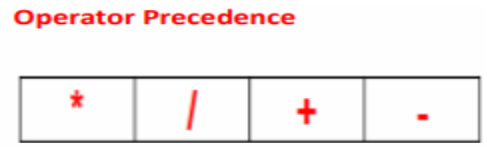
- **Operator precedence+**

Precedence is **the order in which Database software evaluates different operators in the same expression**.

When evaluating an expression containing multiple operators, it evaluates operators with higher precedence before evaluating those with lower precedence

In general, the operators' precedence follows the same rules as in the high school math. The order of the precedence is indicated in the following table.

Operator	Precedence
Unary operators, bitwise NOT (MS SQL Server only)	1
Multiplication and division	2
Addition, subtraction, and concatenation	3
SQL conditions	4



- ✓ Multiplication and division take priority over addition and subtraction.
- ✓ Operators of the same priority are evaluated from left to right.
- ✓ Parentheses are used to force prioritized evaluation and to clarify statements.

3.2. Application of DDL commands

- ➢ CREATE command

Application of " Create database and Create table" commands

a) Create database

i) To create a new database, the SQL query used is CREATE DATABASE

The Syntax is:

Create database database-name;

Always database name should be unique within the RDBMS.

Example of a query to create a database called SCHOOL, In MYSQL, it will look like the following:

```
mysql> create database school;
```

Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use one of these statement:

```
mysql> connect school;
Connection id:    11
Current database: school
mysql> use school;
Database changed
```

b) Create table

Creating the database is the easy part, but at this point it is empty if no table created, as SHOW TABLES tells us:

```
mysql> show tables;
Empty set (0.00 sec)
```

After creating a database and entering in database, there is a need now to create a table. Creating a table involves naming the table and defining its columns and each column's data type. The SQL **CREATE TABLE** statement is used to create a new table.

The basic syntax of CREATE TABLE statement is as follows:

```
Create table table_name (column1 datatype, column2 datatype, column3 datatype,
.....column datatype);
```

Then in brackets comes the list defining each column in the table and what sort of data type it is.

The following SQL query creates a "**Customers**" table with **6 columns**, and thereafter, when the table was successfully created, the message "***Query OK***, o rows affected (0.48 sec)" is displayed.

See below.

```
mysql> CREATE TABLE CUSTOMERS (CID INT, NAME VARCHAR(30), GENDER CHAR, AGE INT, SALARY INT, ADDRESS VARCHAR(20));
Query OK, 0 rows affected (0.11 sec)

mysql>
```

The user can verify if the table has been created successfully by looking at the message displayed by the SQL server, otherwise he/she can use DESC command as follows:

```
mysql> DESC CUSTOMERS;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| CID     | int(11)     | YES  |     | NULL    |       |
| NAME    | varchar(30) | YES  |     | NULL    |       |
| GENDER  | char(1)     | YES  |     | NULL    |       |
| AGE     | int(11)     | YES  |     | NULL    |       |
| SALARY  | int(11)     | YES  |     | NULL    |       |
| ADDRESS | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
6 rows in set (0.09 sec)
```

Now, “Customers” table is created and available in database. It can be used to store required information related to “Customers”. Notice that DESC is the same as DESCRIBE.

Create Table Using another Table: A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. The new table has the same column definitions. All columns or specific columns can be selected.

When you create a new table using existing table, new table would be populated using existing values in the old table. The basic syntax for creating a table from another table is as follows:

```
create table new_table_name as like existing_table_name [ where ]
```

Example: To create a table called SALARY having the same attributes like table “Customers”, write Create table salary like “Customers”;

```
mysql> create table Salary like customers;
Query OK, 0 rows affected (0.16 sec)

mysql>
```

The structure of SALARY is displayed in the following interface.

```
mysql> DESC Salary;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| CID     | int(11)     | YES  |     | NULL    |       |
| NAME    | varchar(30) | YES  |     | NULL    |       |
| GENDER  | char(1)     | YES  |     | NULL    |       |
| AGE     | int(11)     | YES  |     | NULL    |       |
| SALARY  | int(11)     | YES  |     | NULL    |       |
| ADDRESS | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql>
```

Description of SQL constraints:

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints can be either column level or table level. Column level constraints are applied only to one column whereas table level constraints are applied to the entire table.

Constraints are:

NO	Constraint	Description
1	**Primary key constraint**	constraint uniquely identifies each record in a table Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
2	**Foreign key constraint**	Uniquely identifies a row/record in any another database table
3	**Unique key constraint**	Ensures that all the values in a column are different.
4	**Not null constraint**	Ensures that a column cannot have a NULL value.
5	**Default constraint**	Provides a default value for a column when none is specified.
6	**Check constraint**	The CHECK constraint ensures that all values in a column satisfy certain conditions.

- **SQL - ALTER TABLE Command**

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

You should also use the ALTER TABLE command to add and drop various constraints on an existing table.

Below is table of some syntax for SQL ALTER TABLE:

	ADD	DROP
Primary key constraint	ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);	ALTER TABLE table_name DROP CONSTRAINT MyPrimaryKey; Or in MySQL ALTER TABLE table_name DROP PRIMARY KEY;
Unique key constraint	ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);	ALTERTABLEtablename DROPCONSTRAINT UC_tablename;
Not null constraint	ALTER TABLE table_name MODIFY column_name datatype NOT NULL;	
Check constraint	ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);	ALTERTABLEtable_name DROPCONSTRAINT CHK_Columnname;
column	ALTER TABLE table_name ADD column_name datatype;	ALTER TABLE table_name DROP COLUMN column_name;
Data type	ALTER TABLE table_name MODIFY COLUMN column_name datatype;	NOT APPLICABLE

- **Execution of SQL constraints:**

A. Add/drop primary key constraint

A.1. SQL PRIMARY KEY on create table

PRIMARY KEY CONSTRAINT		
MySQL	**MYSQL,SQL Server / Oracle**	**MySQL/SQL Server / Oracle**
PRIMARY KEY constraint on single column		**PRIMARY KEY constraint on multiple columns**
CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, PRIMARY KEY (ID));	CREATE TABLE Persons (ID int NOT NULL PRIMARY KEY, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int);	CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255),Age int, CONSTRAINT PK_Person PRIMARY KEY (ID, LastName));

Note: In the example above at the right, there is only ONE PRIMARY KEY (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

A.2.SQL PRIMARY KEY on ALTER TABLE

The following commands are used when the table is already created.

ADD	
MySQL / SQL Server / Oracle On single column	**MySQL / SQL Server / Oracle On multiple columns**
ALTER TABLE Persons ADD PRIMARY KEY (ID);	ALTER TABLE Persons ADD CONSTRAINT PK_Person PRIMARY KEY (ID, LastName);

Note: If you use ALTER TABLE to add a primary key, the primary key column(s) must have been declared to not contain NULL values (when the table was first created).

A.3.DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

DROP	
MYSQL	**SQL Server / Oracle**
`ALTER TABLE Persons` `DROP PRIMARY KEY;`	`ALTER TABLE Persons DROP` `CONSTRAINT PK_Person;`

B. Add/drop foreign key constraint

The **FOREIGN KEY** constraint is used to prevent actions that would destroy links between tables.

A **FOREIGN KEY** is a field (or collection of fields) in one table, that refers to the **PRIMARY KEY** in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the `PRIMARY KEY` in the "Persons" table.
The "PersonID" column in the "Orders" table is a `FOREIGN KEY` in the "Orders" table.

The `FOREIGN KEY` constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

B.1 SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a `FOREIGN KEY` on the "PersonID" column when the "Orders" table is created:

- **In MySQL:**

CREATE TABLE Orders (OrderID int NOT NULL,OrderNumber int NOT NULL,PersonID int, PRIMARY KEY (OrderID), FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));

- **In MYSQL,SQL Server / Oracle**

CREATE TABLE Orders (OrderID int NOT NULL PRIMARY KEY, OrderNumber int NOT NULL,
PersonID int FOREIGN KEY REFERENCES Persons(PersonID));

B.2 SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

- **MySQL / SQL Server / Oracle**

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

B.3 DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

- **MySQL:**

ALTER TABLE Orders DROP FOREIGN KEY FK_PersonOrder;

- **SQL Server / Oracle**

ALTER TABLE Orders DROP CONSTRAINT FK_PersonOrder;

C. Add/drop unique key constraint

The `UNIQUE` constraint ensures that all values in a column are different.

Both the `UNIQUE` and `PRIMARY KEY` constraints provide a guarantee for uniqueness for a column or set of columns.

A `PRIMARY KEY` constraint automatically has a `UNIQUE` constraint.

Primary key will not accept NULL values whereas **Unique key can accept NULL values**. A table can have only one primary key whereas there can be multiple unique key on a table.

C.1 SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

- **SQL Server / Oracle / MS Access:**

```
CREATE TABLE Persons (ID int NOT NULL UNIQUE, LastName
varchar(255) NOT NULL,
FirstName varchar(255), Age int );
```

- **MySQL:**

```
CREATE TABLE Persons ( ID int NOT NULL, LastName varchar(255) NOT NULL,
FirstName varchar(255), Age int, UNIQUE (ID));
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following

SQL syntax:

- **MySQL / SQL Server / Oracle**

```
CREATE TABLE Persons ( ID int NOT NULL, LastName varchar(255) NOT NULL,
FirstName varchar(255), Age int, CONSTRAINT UC_Person UNIQUE (ID,LastName));
```

C2. SQL UNIQUE Constraint on ALTER TABLE

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

- **MySQL / SQL Server / Oracle / MS Access:**

```
ALTER TABLE Persons ADD UNIQUE (ID);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

- **MySQL / SQL Server / Oracle:**

```
ALTER TABLE Persons ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

C.3 DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

- **MySQL:**

ALTER TABLE Persons DROP INDEX UC_Person;

- **SQL Server / Oracle / MS Access:**
  ALTER TABLE Persons DROP CONSTRAINT UC_Person;

D. add /drop not null constraint

By default, a column can hold NULL values.
The `NOT NULL` constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

D1. SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:
Example:

CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255) NOT NULL,Age int);

D2. SQL NOT NULL on ALTER TABLE

To create a `NOT NULL` constraint on the "Age" column when the "Persons" table is already created, use the following SQL:
ALTER TABLE Persons MODIFY Age int NOT NULL;

D3. DROP NOT NULL CONSTRAINT

In this tutorial, we are going to see how to drop not null constraint in MySQL. To remove the constraint **NOT NULL** for a column in MySQL, use the statement **ALTER TABLE … MODIFY** and reformulate the column definition by removing the **NOT NULL** attribute.

Syntax:
Here is the syntax to remove the constraint **NOT NULL** for a column in MySQL

ALTER TABLE table_Name MODIFY columnName DATA_TYPE;

Example :
Here is an example to remove the constraint **NOT NULL** for the column “Name” of the “Employee” table:

```
ALTER TABLE Employee MODIFY Name VARCHAR (20);
```

To make sure you don’t miss anything, you can use the statement **SHOW CREATE TABLE** to display the full definition of the column:

```
SHOW CREATE TABLE Employee;
```

E. Add/drop 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.

E1. SQL DEFAULT on CREATE TABLE

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

My SQL / SQL Server / Oracle

```
CREATE TABLE Perss ( ID int NOT NULL, LastName varchar(40) NOT NULL, FirstName varchar(20),
Age int, City varchar(25) DEFAULT 'Sandnes');
```

E2. SQL DEFAULT on ALTER TABLE

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

- **MySQL:**

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

- **SQL Server:**

```
ALTER TABLE Persons ADD CONSTRAINT df_City DEFAULT 'Sandnes' FOR City;
```

- **Oracle:**

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

E3. DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

- **MySQL:**

```
ALTER TABLE Persons ALTER City DROP DEFAULT;
```

- **SQL Server / Oracle:**

```
ALTER TABLE Persons ALTER COLUMN City DROP DEFAULT;
```

F. Add/drop check constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

F1. SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

- **MySQL:**

```
CREATE TABLE P (ID int NOT NULL,LastName varchar(255) NOT NULL,FirstName varchar(255),Age int, CHECK (Age>=18));
```

- **SQL Server / Oracle**

```
CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL,FirstName varchar(255),Age int CHECK (Age>=18));
```

NOTE: To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

- **MySQL / SQL Server / Oracle**

```
CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL,FirstName varchar(255),Age int,City varchar(255),
CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes'));
```

F2.SQL CHECK on ALTER TABLE

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

- **MySQL / SQL Server / Oracle**

```
ALTER TABLE Persons ADD CHECK (Age>=18);
```

NOTE: To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

- **MySQL / SQL Server**

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
```

F3. DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL

- **MySQL:**

```
ALTER TABLE Persons DROP CHECK CHK_PersonAge;
```

- **SQL Server / Oracle**

```
ALTER TABLE Persons DROP CONSTRAINT CHK_PersonAge;
```

➢ ALTER Table

ALTER TABLE Statement

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:

Example

```
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:

Example

```
ALTER TABLE Customers DROP COLUMN Email;
```

ALTER TABLE - MODIFY COLUMN

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

ALTER TABLE *table_name* MODIFY COLUMN *column_name datatype*;

- **DROP**

This command deletes the information of the table or the database, and it removes the entire structure/schema of the table or the entire database.

By Using the DROP statement, the objects are permanently deleted or lost from a database, and they cannot be rolled back.

Whenever an object is deleted, its description is deleted from the catalog, and any packages that reference the object are invalidated.

- **Database**

The DROP DATABASE statement is used to drop an existing SQL database.

Syntax:
DROP DATABASE *databasename*;

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

DROP DATABASE Example
The following SQL statement drops the existing database "testDB":
Example
DROP DATABASE testDB;

Tip: Once a database is dropped, you can check it in the list of databases with the following SQL command:

SHOW DATABASES;

- **Table**

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

Syntax

DROP TABLE *table_name*;

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

MySQL DROP TABLE Example

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

Example

```
DROP TABLE Shippers;
```

➢ TRUNCATE Table

The TRUNCATE statement in MySQL removes the complete data without removing its structure. It is a part of **DDL or data definition language command**. Generally, we use this command when we want to delete an entire data from a table without removing the table structure.

The TRUNCATE command works the same as a DELETE command without using a **WHERE clause** that deletes complete rows from a table. However, the TRUNCATE command is more efficient as compared to the **DELETE** command because it removes and recreates the table instead of deleting single records one at a time.

The following points must be considered while using the TRUNCATE command:

- ✓ We cannot use the **WHERE** clause with this command so that filtering of records is not possible.
- ✓ We **cannot rollback the deleted data** after executing this command because the log is not maintained while performing this operation.
- ✓ We cannot use the truncate statement when a table is referenced by a **foreign key** or participates in an **indexed view**.
- ✓ The TRUNCATE command doesn't fire DELETE **triggers** associated with the table that is being truncated because it does not operate on individual rows.

Syntax

The following syntax explains the TRUNCATE command to remove data from the table:

```
TRUNCATE [TABLE] table_name;
```

Example:

```
mysql> TRUNCATE TABLE customer;
```

➢ **MODIFY**

- **Database**

SQL ALTER DATABASE is an essential statement to modify the properties of an existing database in a Relational Database Management System (RDBMS). This statement is helpful if you want to rename a database, change its composition, or modify file properties.

- **Table**

The **modify** command is used when we have to modify a column in the existing table, like adding a new one, modifying the datatype for a column, and dropping an existing column.

By using this command, we have to apply some changes to the result set field.

) Add a new column to the existing table

ALTER TABLE and **ADD COLUMN** statement is used to add one or more new columns to an existing table.

a) **Add a single column to a table.**

Syntax:

```
ALTER TABLE <table_name> ADD COLUMN <new_column_name> <Datatype>
[CONSTRAINTS];
```

Example

```
ALTER TABLE csharpcorner_mvps ADD COLUMN MVPKitStatus INT;
```

Note: Use the "DESCRIBE" command to check the result from the ALTER TABLE ADD statement

b) Add multiples column to a table.

Syntax

```
ALTER TABLE employees
ADD COLUMN date_of_birth DATE NOT NULL,
ADD COLUMN department VARCHAR(50) DEFAULT 'IT',
ADD COLUMN salary DECIMAL(10,2) NOT NULL;
```

Example.

```
ALTER TABLE csharpcorner_mvps
ADD COLUMN Country VARCHAR(50),
ADD COLUMN Description VARCHAR(250);
```

Note. Use the "DESCRIBE" command to check the result from the ALTER TABLE ADD statement.

ALTER TABLE MODIFY statement is used to modify one or more columns to an existing table.

a) Modify a single column in a table

Syntax

```
ALTER TABLE employees
MODIFY employee_name VARCHAR(100);
```

Example.

```
ALTER TABLE csharpcorner_mvps
MODIFY MVPKitStatus VARCHAR(100);
```

Note.

Use the “DESCRIBE” command to check the result from the ALTER TABLE MODIFY statement.

b) Modify multiple columns in a table

Syntax

```
ALTER TABLE employees
MODIFY employee_name VARCHAR(100),
MODIFY salary DECIMAL(10,2);
```

Example.

```
ALTER TABLE csharpcorner_mvps
MODIFY MVPAddress VARCHAR(250) NOT NULL,
MODIFY MVPKitStatus VARCHAR(100) NOT NULL,
MODIFY Description VARCHAR(221);
```

SQL

Copy

Note.

Use the “DESCRIBE” command to check the result from the ALTER TABLE MODIFY statement.

3) DROP a Column

ALTER TABLE DROP COLUMN statement is used to drop the column(s) in a table.

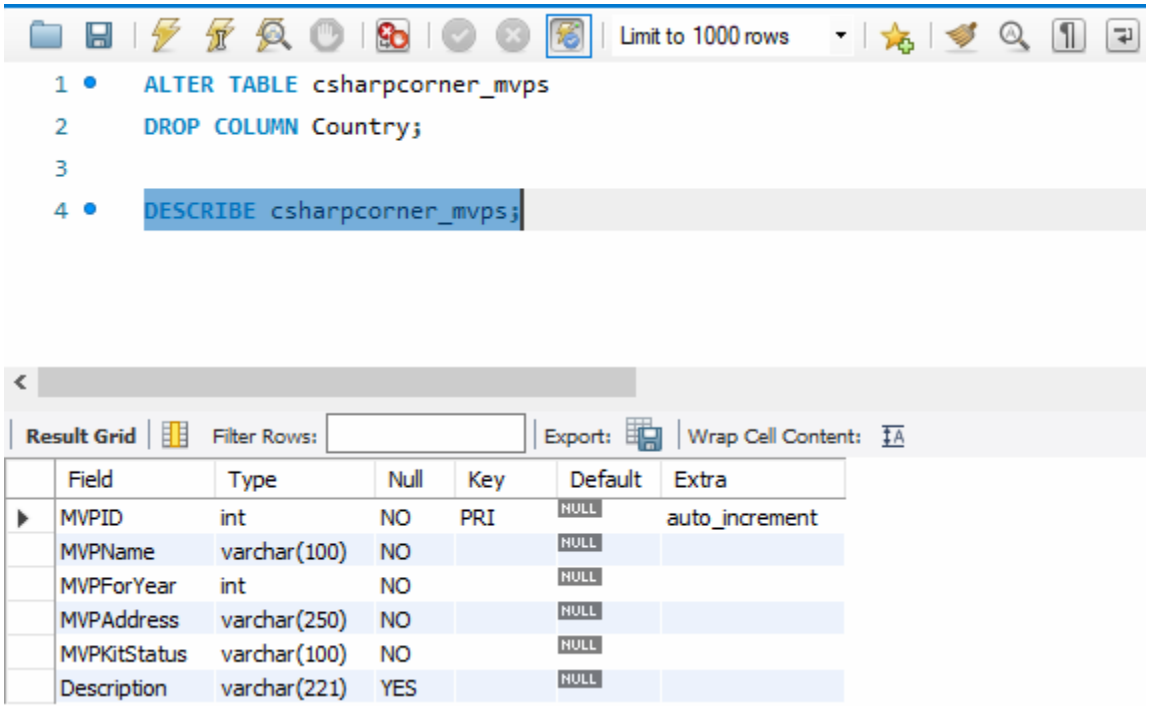
Syntax

```
ALTER TABLE employees DROP COLUMN date_of_birth;
```

Example.

```
ALTER TABLE csharpcorner_mvps DROP COLUMN Country;
```

Note. Use the "DESCRIBE" command to check the result from the ALTER TABLE DROP statement.



```
ALTER TABLE csharpcorner_mvps
DROP COLUMN Country;

DESCRIBE csharpcorner_mvps;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
MVPID	int	NO	PRI	NULL	auto_increment
MVPName	varchar(100)	NO		NULL	
MVPForYear	int	NO		NULL	
MVPAddress	varchar(250)	NO		NULL	
MVPKitStatus	varchar(100)	NO		NULL	
Description	varchar(221)	YES		NULL	

4) Change Column name

To rename a column in a table, use the following statement.

Syntax

```
ALTER TABLE employees CHANGE COLUMN emp_name employee_name VARCHAR(100);
```

Example.

```
ALTER TABLE STUDENT CHANGE COLUMN SEX GENDER CHAR;
```

Note. Use the "DESCRIBE" command to check the result from the ALTER TABLE CHANGE statement.

3.3 Application of DML commands

- **INSERT**

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax:

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...) VALUES (*value1*, *value2*, *value3*, ...);

Example:
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query.

However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table_name* VALUES (*value1*, *value2*, *value3*, ...);

Example:
INSERT INTO Customers VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

Insert Data Only in Specified Columns

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

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example

INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway');

- **UPDATE**

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

UPDATE Syntax:

UPDATE *table_name* SET *column1* = *value1*, *column2* = *value2*, ... WHERE *condition*;

Note: Be careful when updating records in a table!

The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example

```
UPDATE Customers SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'
WHERE CustomerID = 1;
```

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the PostalCode to 00000 for all records where country is "Mexico":

Example

```
UPDATE Customers SET PostalCode = 00000 WHERE Country = 'Mexico';
```

- **DELETE**

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

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement.

The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

SQL DELETE Example:

The following SQL statement deletes the customer "**Alfreds Futterkiste**" from the "Customers"

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

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*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

DELETE FROM Customers;

➢ CALL

Stored procedures are sub routines, segment of SQL statements which are stored in SQL catalog. These procedures contain IN and OUT parameters, or both. They may return result sets in case you use SELECT statements; they can return multiple result-sets.

✓ Syntax in MYSQL

```
DELIMITER
CREATE PROCEDURE ProcedureName()
BEGIN
SQL STATEMENT
END
DELIMITER
```

EXAMPLE

Consider the table BOOKS below:

```
+--------+-------------+------+------------+----------------+--------+
| BookID | Bookname    | ed2  | Author     | Published_date | Number |
+--------+-------------+------+------------+----------------+--------+
| B001   | Web design  | Ed2  | H. Olivier |           1997 |     30 |
| B002   | Database    | Ed1  | P. Albert  |           2001 |     20 |
| B003   | VB          | Ed3  | M. Claude  |           1985 |     14 |
| B004   | Web design  | Ed1  | J. Mata    |           1998 |     26 |
| B005   | Programming | Ed1  | G.John     |           2003 |     25 |
+--------+-------------+------+------------+----------------+--------+
```

Now we are going to create a procedure and call it mine.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE mine()
    -> BEGIN
    -> SELECT * FROM BOOKS;
    -> END
    -> //
Query OK, 0 rows affected (0.20 sec)
```

The call statement of MySQL is used to invoke/call a stored procedure.

Syntax

Following is the syntax of the CALL statement in MySQL

CALL procedure_name(parameter[param1, param2, ...])

Where procedure_name is the name of an existing procedure you need to call and param1 and param2... are the list of parameters accepted by the procedure (if it accepts any).

The CALL statement invokes a stored procedure that was defined previously with CREATE PROCEDURE . Stored procedures that take no arguments can be invoked without parentheses.

Calling a stored procedure with parameters

Suppose we have created a table named **Emp** in the database using the CREATE statement and inserted three records in it as shown below −

```
CREATE TABLE Emp (
  Name VARCHAR(255),
  Salary INT,
  Location VARCHAR(255));
```

Assume we have created a stored procedure InsertData which accepts the name, salary and location values and inserts them as a record into the above create (Emp) table.

```
DELIMITER //
Create procedure InsertData (
  IN name VARCHAR(30),
  IN sal INT,
  IN loc VARCHAR(45))
    BEGIN
      INSERT INTO Emp(Name, Salary, Location) VALUES (name, sal, loc);
    END //
DELIMITER ;
```

Following statement calls the above created stored procedure

```
CALL InsertData ('Raju', 35000, 'Bangalore');
```

```
CALL InsertData ('Raman', 45000, 'Vishakhapatnam');
CALL InsertData ('Rahman', 55000, 'Hyderabad');
```

Verification
Once you call the procedure by passing the required values you can verify the contents of the Emp table as shown below −

```
SELECT * FROM EMP;
```

The above query produces the following output −

Name	Salary	Location
Raju	35000	Bangalore
Raman	45000	Visakhapatnam
Rahman	55000	Hyderabad

Calling a stored procedure without parameters

While calling a stored procedure that doesn't accepts any arguments, we can omit the parenthesis as shown below:

```
CALL procedure;
```

Assume we have created another procedure with name getData that retries the contents of the table EMP

```
DELIMITER //
CREATE PROCEDURE getData()
  BEGIN
    SELECT * FROM EMP;
  END//
DELIMITER ;
```

Since this procedure doesn't accept arguments you can call this procedure by omitting the parameters as shown below

```
CALL getData;
```

Output
Following is the output of the above query

Name	Salary	Location

Raju	35000	Bangalore
Raman	45000	Visakhapatnam
Rahman	55000	Hyderabad

- **LOCK**

MySQL LOCK TABLES Statement
The following is the syntax that allows us to acquire a table lock explicitly:

LOCK TABLES table_name [READ | WRITE];

In the above syntax, we have specified the **table name** on which we want to acquire a lock after the **LOCK TABLES** keywords. We can specify the **lock type**, either READ or WRITE.

We can also lock more than one table in MySQL by using a list of comma-separated table's names with lock types. See the below syntax:

LOCK TABLES tab_name1 [**READ** | WRITE], tab_name2 [**READ** | WRITE],..... ;

Example:

LOCK TABLE info_table READ;

Write Locks
The following are the features of a WRITE lock:

- It is the session that holds the lock of a table and can read and write data both from the table.
- It is the only session that accesses the table by holding a lock. And all other sessions cannot access the data of the table until the WRITE lock is released.

mysql> LOCK **TABLE** info_table WRITE;

MySQL UNLOCK TABLES Statement

The following is the syntax that allows us **to release a lock** for a table in MySQL:

mysql> **UNLOCK TABLES;**

3.4 Application of DQL Command

- **SELECT**

The SELECT statement in MySQL is used to **fetch data from one or more tables**. We can retrieve records of all fields or specified fields that match specified criteria using this statement.

SELECT Statement Syntax:

The general syntax of this statement to fetch data from tables are as follows:
SELECT field_name1, field_name 2,..., field_nameN **FROM** table_name1, **WHERE** condition;

Syntax for all fields:

SELECT * **FROM** tables **WHERE** conditions;

The SELECT statement uses the following parameters:

Parameter Name	Descriptions
field_name(s) or *	It is used to specify one or more columns to returns in the result set. The asterisk (*) returns all fields of a table.
table_name(s)	It is the name of tables from which we want to fetch data.
WHERE	It is an optional clause. It specifies the condition that returned the matched records in the result set.

Example:

1. If we want to retrieve a **single column from the table**, we need to execute the below query:

SELECT Name FROM employee_detail;

2. If we want to query **multiple columns from the table**, we need to execute the below query:

SELECT Name, Email, City **FROM** employee_detail;

3. If we want to fetch data from **all columns of the table**, we need to use all column's names with the select statement. Specifying all column names is not convenient to the user, so MySQL uses an **asterisk** (*) to retrieve all column data as follows:

SELECT * **FROM** employee_detail;

The MySQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

SELECT DISTINCT column1, column2, ... FROM table_name;

SELECT Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "Country" column in the "Customers" table:

Example

```
SELECT Country FROM Customers;
```

Now, let us use the SELECT DISTINCT statement and see the result.

SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

Example

```
SELECT DISTINCT Country FROM Customers;
```

The following SQL statement counts and returns the number of different (distinct) countries in the "Customers" table:

Example

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

➢ **SQL aggregate function**

An aggregate function in **SQL** performs a calculation on multiple values and returns a single value.

An aggregate function in SQL returns one value after calculating multiple values of a column. We often use aggregate functions with the **GROUP BY** and **HAVING clauses** of the SELECT statement.

An aggregate function ignores NULL values when it performs the calculation, except for the count function.

There are 5 types of SQL aggregate functions:

- ✓ **AVG**: calculates the average of a set of values.
- ✓ **SUM**: calculates the sum of values.
- ✓ **MIN**: gets the minimum value in a set of values.
- ✓ **MAX**: gets the maximum value in a set of values.
- ✓ **COUNT**: counts rows in a specified table or view.

Example:

Consider the following STUDENTS **table**

STUDID	STUD_NAME	STUD_CLASS	STUD_AGE	STUD_SEX
S01	BENIMANA Consolee	L5SOD	19	FEMALE
S02	BIZIMANA JMV	L5SOD	20	MALE
S03	BYISHIMO EDISON	L5SOD	20	MALE
S04	DUSHIMIMANA CLEMENT	L5SOD	21	MALE
S05	DUSINGIZEMARIYA JOSELYNE	L5SOD	19	FEMALE

Function	Syntax	Example
COUNT ()	SELECT COUNT (*column_name*) FROM *table_name* WHERE *condition*;	SELECTCOUNT(*)ASCOUNT_STUD_AGEFROMSTUDENTS; SELECTCOUNT(STUD_AGE)ASCOUNT_STUD_AGEFROMSTUDENTS; Result:5
AVG ()	SELECT AVG (*column_name*) FROM *table_name* WHERE *condition*;	SELECTAVG(STUD_AGE) AS AVG_STUD_AGEFROMSTUDENTS; Result: 19
SUM ()	SELECT SUM (*column_name*) FROM *table_name* WHERE *condition*;	SELECTSUM(STUD_AGE)ASSUM_STUD_AGEFROMSTUDENTS; Result:99
MIN ()	SELECT MIN (*column_name*) FROM *table_name* WHERE *condition*;	SELECTMIN(STUD_AGE)ASMIN_STUD_AGEFROMSTUDENTS; Result:19

➢ **SQL clause**

- SQL clause helps us to retrieve a set or bundles of records from the table.
- SQL clause helps us to specify a condition on the columns or the records of a table.

Different clauses available in the Structured Query Language are as follows:

1. WHERE CLAUSE
2. GROUP BY CLAUSE
3. HAVING CLAUSE
4. ORDER BY CLAUSE

1. WHERE CLAUSE

A WHERE clause in SQL is used with the SELECT query, which is one of the data manipulation language commands. WHERE clauses can be used to limit the number of rows to be displayed in the result set, it generally helps in filtering the records. It returns only those queries which fulfill the specific conditions of the WHERE clause. WHERE clause is used in SELECT, UPDATE, DELETE statement, etc.

WHERE clause with SELECT Query

Asterisk symbol is used with a WHERE clause in a SELECT query to retrieve all the column values for every record from a table.

Syntax of where clause with a select query to retrieve all the column values for every record from a table:

1. **SELECT * FROM** TABLENAME **WHERE** CONDITION;

Example 1:

Write a query to retrieve all those records of an employee where employee salary is greater than 50000.

Query: mysql> **SELECT * FROM** employees **WHERE** Salary > 50000;

The above query will display all those records of an employee where an employee's salary is greater than 50000. Below 50000 salary will not be displayed as per the conditions.

Example 2:

Write a query to update the employee's record and set the updated name as 'Harshada Sharma' where the employee's city name is Jaipur.

Query:

mysql> UPDATE employees SET Name = "Harshada Sharma" WHERE City = "Jaipur";

The above query will update the employee's name to "Harshada Sharma," where the employee's city is Jaipur.

To verify whether records are updated or not, we will run a select query.

mysql> **SELECT * FROM** employees;

Example 3:

Write a query to delete an employee's record where the employee's joining date is "2013-1212".

Query:

mysql> **DELETE FROM** employees **WHERE** Date_of_Joining = "2013-12-12";

The above query will delete the employee details of the employee whose joining date is "2013-12-12".

To verify the results of the above query, we will execute the select query.

mysql> **SELECT *FROM** employees;

2. GROUP BY CLAUSE

The Group By clause is used to arrange similar kinds of records into the groups in the Structured Query Language. The Group by clause in the Structured Query Language is used with Select Statement. Group by clause is placed after the where clause in the SQL statement. The Group By clause is specially used with the aggregate function, i.e., max (), min (), avg (), sum (), count () to group the result based on one or more than one column.

The syntax of Group By clause:

SELECT * FROM TABLENAME **GROUP BY** COLUMNNAME;

The above syntax will select all the data or records from the table, but it will arrange all those data or records in the groups based on the column name given in the query.

The syntax of Group By clause with Aggregate Functions:

SELECT COLUMNNAME1, Aggregate_FUNCTION (COLUMNNAME) **FROM** TABLE NAME **GROUP BY** COLUMNNAME;

Let's understand the Group By clause with the help of examples.

Example 1:

Write a query to display all the records of the employees table but group the results based on the age column.

Query:

mysql> **SELECT * FROM** employees **GROUP BY** Age;

The above query will display all the records of the employees table but grouped by the age column.

Example 2:

Write a query to display all the records of the employees table grouped by the designation and salary.

Query: mysql> **SELECT * FROM** employees **GROUP BY** Salary, Designation;

The above query will display all the records of the employees table but grouped by the salary and designation column.

Examples of Group By clause using aggregate functions

Example 1:

Write a query to list the number of employees working on a particular designation and group the results by designation of the employee.

Query:

mysql> **SELECT** COUNT (E_ID) **AS** Number_of_Employees, Designation **FROM** employ ees **GROUP BY** Designation;

The above query will display the designation with the respective number of employees working on that designation. All these results will be grouped by the designation column.

Example 2:

Write a query to display the sum of an employee's salary as per the city grouped by an employee's age.

Query: mysql> **SELECT** SUM (Salary) **AS** Salary, City **FROM** employees **GROUP BY** City;

The above query will first calculate the sum of salaries working in each city, and then it will display the salary sum with the respective salary but grouped by the age column.

3. HAVING CLAUSE

When we need to place any conditions on the table's column, we use the WHERE clause in SQL. But if we want to use any condition on a column in Group By clause at that time, we will use the HAVING clause with the Group By clause for column conditions.

Syntax:

TABLENAME **GROUP BY** COLUMNNAME **HAVING** CONDITION;

Example 1:

Write a query to display the name of employees, salary, and city where the employee's maximum salary is greater than 40000 and group the results by designation.

Query:

mysql> **SELECT Name**, City, **MAX** (Salary) **AS** Salary **FROM** employees **GROUP BY** D esignation **HAVING MAX** (Salary) > 40000;

Example 2:

mysql> **SELECT Name**, Designation, SUM (Salary) **AS** Salary **FROM** employees **GROUP BY** City **HAVING** SUM (Salary) > 45000;

4. ORDER BY CLAUSE

Whenever we want to sort anything in SQL, we use the ORDER BY clause.

The ORDER BY clause in SQL will help us to sort the data based on the specific column of a table.

As we all know, sorting means either in ASCENDING ORDER or DESCENDING ORDER. In the same way, ORDER BY CLAUSE sorts the data in ascending or descending order as per our

requirement. The data will be sorted in ascending order whenever the ASC keyword is used with ORDER by clause, and the DESC keyword will sort the records in descending order.

By default, sorting in the SQL will be done using the ORDER BY clause in ASCENDING order if we didn't mention the sorting order.

Syntax of ORDER BY clause without asc and desc keyword:

```
SELECT COLUMN_NAME1, COLUMN_NAME2 FROM TABLE_NAME ORDER BY
COLUMNAME;
```

Syntax of ORDER BY clause to sort in ascending order:

```
SELECT COLUMN_NAME1, COLUMN_NAME2 FROM TABLE_NAME ORDER BY
COLUMN_NAME ASC;
```

Syntax of ORDER BY clause to sort in descending order:

```
SELECT COLUMN_NAME1, COLUMN_NAME2 FROM TABLE_NAME ORDER BY
COLUMN_NAME DESC;
```

Example 1:

Write a query to sort the records in the ascending order of the employee designation from the employees table.

Query: mysql> **SELECT * FROM** employees **ORDER BY**
Designation;

Here in a SELECT query, an ORDER BY clause is applied on the column 'Designation' to sort the records, but we didn't use the ASC keyword after the ORDER BY clause to sort in ascending order. So, by default, data will be sorted in ascending order if we don't specify asc keyword.

Example 2:

Write a query to display employee name and salary in the ascending order of the employee's salary from the employees table.

Query:

mysql> **SELECT Name**, Salary **FROM** employees **ORDER BY** Salary **ASC**;

Here in a SELECT query, an ORDER BY clause is applied to the 'Salary' column to sort the records. We have used the ASC keyword to sort the employee's salary in ascending order.

Example 3:

Write a query to sort the data in descending order of the employee name stored in the employees table.

Query:

mysql> **SELECT * FROM** employees **ORDER BY Name DESC**;

Here we have used the ORDER BY clause with the SELECT query applied on the Name column to sort the data. We have used the DESC keyword after the ORDER BY clause to sort data in descending order.

Application of DCL commands

DCL commands are used to manage database security and access control.

The two primary DCL commands are:
• **GRANT**: Used to grant specific privileges to database users or roles.
• **REVOKE**: Used to revoke previously granted privileges.

Some common privileges include:

- **`ALL PRIVILEGES`:** The user is granted all privileges except GRANT OPTION and PROXY.
- **`ALTER`:** The user can change the structure of a table or database.
- **`CREATE`:** The user can create new databases and tables.
- **`DELETE`:** The user can delete rows in a table.
- **`INSERT`:** The user can add rows to a table.
- **`SELECT`:** The user can read rows from a table.
- **`UPDATE`:** The user can update rows in a table.

➢ GRANT Statement

The grant statement enables system administrators to ***assign privileges and roles*** to the MySQL user accounts so that they can use the assigned permission on the database whenever required.

Syntax
The following are the basic syntax of using the GRANT statement:

GRANT privilege_name(s) **ON** object **TO** user_account_name;

or

```
GRANT privilege_name ON object_name TO user_name WITH GRANT OPTION;
```

Parameter Explanation

In the above syntax, we can have the following parameters:

Parameter Name	Descriptions
privilege_name(s)	It specifies the access rights or grant privilege to user accounts. If we want to give multiple privileges, then use a comma operator to separate them.
object	It determines the privilege level on which the access rights are being granted. It means granting privilege to the table; then the object should be the name of the table.
user_account_name	It determines the account name of the user to whom the access rights would be granted.

Example:

```
GRANT SELECT, INSERT ON Employees TO HR_Manager;
```

This grants the "HR_Manager" role the privileges to select and insert data into the "Employees" table.

- **REVOKE statement**

REVOKE: *The REVOKE command is used to revoke previously granted privileges:*

Syntax

```
REVOKE [privilege_name] ON [object_name] FROM [user_name];
```

```
REVOKE DELETE ON Customers FROM Sales_Team;
```

This revokes the privilege to delete data from the "Customers" table from the "Sales_Team" role.

3.6 Application of TCL commands

- **Database transaction control:**

What is a transaction?

A transaction, in the context of a database, is a logical unit that is independently executed for data retrieval or updates.

A database transaction is ***a series of one or more operations executed as a single atomic unit of work.***

Experts talk about a database transaction as a "unit of work" that is achieved within a database design environment.

The effects of all the SQL statements in a transaction can be either all committed (applied to the database) or all rolled back (undone from the database), insuring data consistency.

Transactions have the following four standard properties, usually referred to by the acronym

ACID.

Property	Description
Atomicity	Ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.
Consistency	Ensures that the database properly changes states upon a successfully committed transaction.
Isolation	Enables transactions to operate independently of and transparent to each other.
Durability	Ensures that the result or effect of a committed transaction persists in case of a system failure.

The following commands are used to control transactions but we are going to focus on the first three.

Command	Description
COMMIT	Used to save the changes
ROLLBACK	Used to roll back the changes
SAVEPOINT	Used to create points within the groups of transactions in which to ROLLBACK.
SET TRANSACTION	Places a name on a transaction.

Transactional control commands are only used with the DML Commands such as **INSERT**, **UPDATE** and **DELETE** only.

They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

➢ COMMIT

The COMMIT Command The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database.

The syntax for the COMMIT command is as follows.

```
COMMIT;
```

Example Consider the CUSTOMER table having the following records.

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

Following is an example which would delete those records from the table which have age = 24 and then **COMMIT** the changes in the database.

```
BEGIN / START TRANSACTION
DELETE FROM CUSTOMER WHERE CAGE= 24;
COMMIT;
```

➢ ROLLBACK

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows

```
ROLLBACK;
```

Example: Consider the CUSTOMER table having the following records

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

Following is an example, which would delete those records from the table which have the age = 24 and then ROLLBACK the changes in the database.

```
BEGIN / START TRANSACTION
DELETE FROM CUSTOMER WHERE CAGE= 24;
ROLLBACK;
```

Thus, the delete operation would not impact the table and the SELECT statement would produce the following result.

CID	CLAST_NAME	CFIRST_NAME	CAGE	CADDRESS	CSALARY
1	KAZE	OLGA	32	KIGALI	2000.00
2	ISHIMWE	NAOME	25	KAMONYI	1500.00
3	ISHIMWE	SAMUEL	23	MUHANGA	2000.00
4	GATETE	YOUSSOUF	25	RUHANGO	6500.00
5	NISHIMWE	ALICE	27	NYANZA	4500.00
6	TUYIZERE	JOSIANE	22	HUYE	4500.00
7	UWIRAGIYE	MONIQUE	24	NYAMAGABE	10000.00

➢ SAVEPOINT

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINT SAVEPOINT_NAME;
```

This command serves only in the creation of a SAVEPOINT among all the transactional statements.

The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below.

```
ROLLBACK TO SAVEPOINT_NAME;
```

Following is an example where you plan to delete the three different records from the CUSTOMER table.

You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

➢ SET Transaction

The SET TRANSACTION Statement in MYSQL is used to set the values to the characteristics of the current transaction such as transaction isolation level and access mode.

Use the SET TRANSACTION statement to establish the current transaction as read-only or read/write, establish its isolation level, assign it to a specified rollback segment, or assign a name to the transaction.

A transaction implicitly begins with any operation that obtains a TX lock:

- When a statement that modifies data is issued
- When a SELECT ... FOR UPDATE statement is issued
- When a transaction is explicitly started with a SET TRANSACTION statement or the DBMS_TRANSACTION package

Issuing either a COMMIT or ROLLBACK statement explicitly ends the current transaction. The operations performed by a SET TRANSACTION statement affect only your current transaction, not other users or other transactions.

Your transaction ends whenever you issue a COMMIT or ROLLBACK statement.

➢ SET Constraints

What is a SQL constraint?
SQL constraints are rules that allow data to be entered into a table only if it meets the predefined conditions.

They are part of your database schema, the broader set of rules that governs your database.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- 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
- CHECK: Ensures that the values in a column satisfies a specific condition
- DEFAULT: Sets a default value for a column if no value is specified
- CREATE INDEX: Used to create and retrieve data from the database very quickly

END OF LO 3