Module 4: Structured Query Language (SQL)

- Overview of SQL
- Data Definition Language Commands
- Integrity constraints: key constraints, Domain Constraints, Referential integrity , check constraints,
- Data Manipulation commands
- Data Control commands
- Set and string operations
- aggregate function-group by, having, Views in SQL, joins, Nested and complex queries
- Triggers(ECA Model)
- Security and Authorization in SQL

- **Overview of SQL**
- SQL is a language to operate databases; it includes Database Creation, Database Deletion, Fetching Data Rows, Modifying & Deleting Data rows, etc.
- **SQL** stands for **Structured Query Language** which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL was developed in the 1970s by IBM Computer Scientists and became a standard of the American National Standards Institute (ANSI) in 1986, and the International Organization for Standardization (ISO) in 1987.

SQL is the standard language to communicate with Relational Database Systems. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their Standard Database Language.

Why SQL?

SQL is widely popular because it offers the following advantages –

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

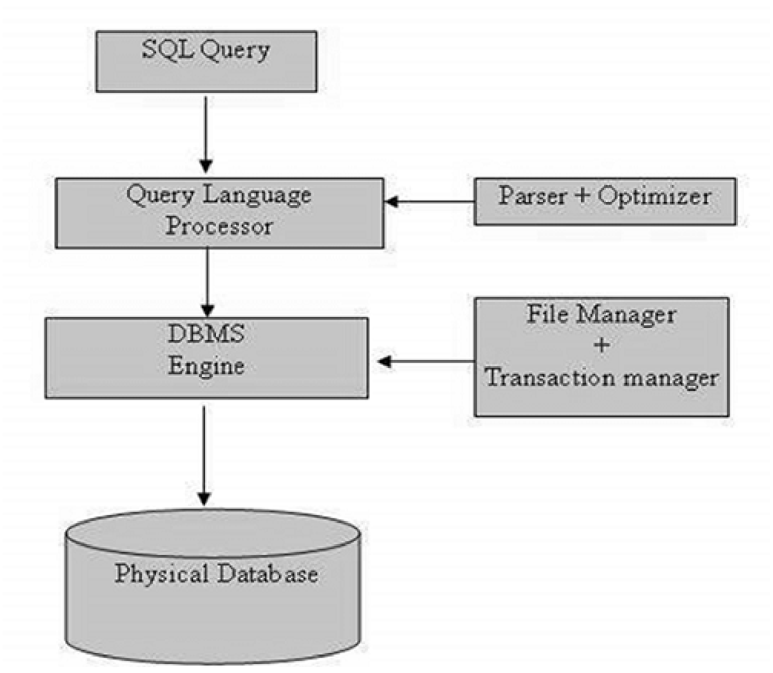
How SQL Works?

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in this process. These components are –

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files. Following is a simple diagram showing the SQL Architecture –



• Data Definition Language Commands

Data Definition Language(DDL) is a subset of SQL and a part of DBMS(Database Management System). DDL consist of Commands to commands like CREATE, ALTER, TRUNCATE and DROP. These commands are used to create or modify the tables in SQL.

DDL Commands:

1. Create
2. Alter
3. truncate
4. drop

Command-1:
CREATE :
This command is used to create a new table in SQL. The user has to give information like table name, column names, and their datatypes.
Syntax –

```
CREATE TABLE table_name

(

column_1 datatype,

column_2 datatype,

column_3 datatype,

....

);
```

Example –

We need to create a table for storing Student information of a particular College. Create syntax would be as below.

```
CREATE TABLE Student_info

(

College_Id number(2),

College_name varchar(30),

Branch varchar(10)

);
```

Command-2:
ALTER :
This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can do add, delete or modify tasks easily.
Syntax –
Syntax to add a column to an existing table.

```
ALTER TABLE table_name

ADD column_name datatype;
```

Example –
In our Student_info table, we want to add a new column for CGPA. The syntax would be as below as follows.

```
ALTER TABLE Student_info

ADD CGPA number;
```

Command-3:
TRUNCATE :
This command is used to remove all rows from the table, but the structure of the table still exists.
Syntax –
Syntax to remove an existing table.

```
TRUNCATE TABLE table_name;
```

Example –
The College Authority wants to remove the details of all students for new batches but wants to keep the table structure. The command they can use is as follows.

```
TRUNCATE TABLE Student_info;
```

Command-4:
DROP :
This command is used to remove an existing table along with its structure from the Database.
Syntax –
Syntax to drop an existing table.

```
DROP TABLE table_name;
```

Example –
If the College Authority wants to change their Database by deleting the Student_info Table.

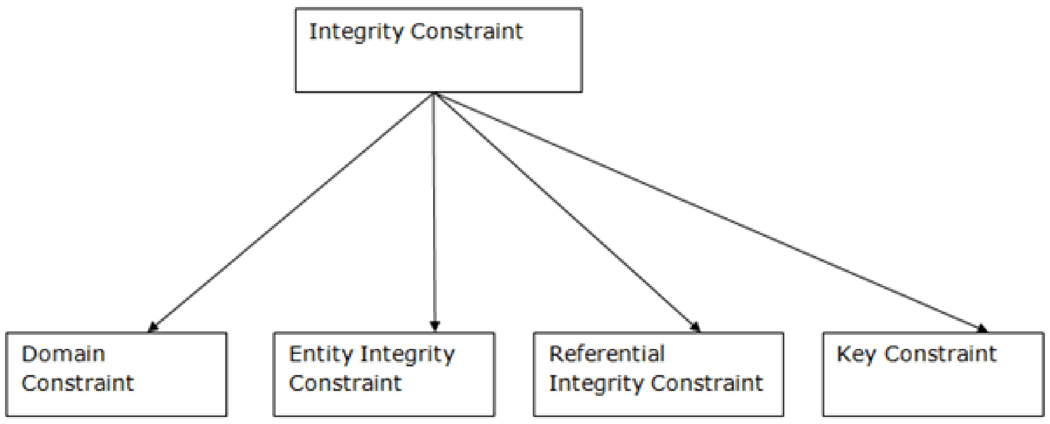
```
DROP TABLE Student_info;
```

- **Integrity constraints: key constraints, Domain Constraints, Referential integrity , check constraints.**

Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

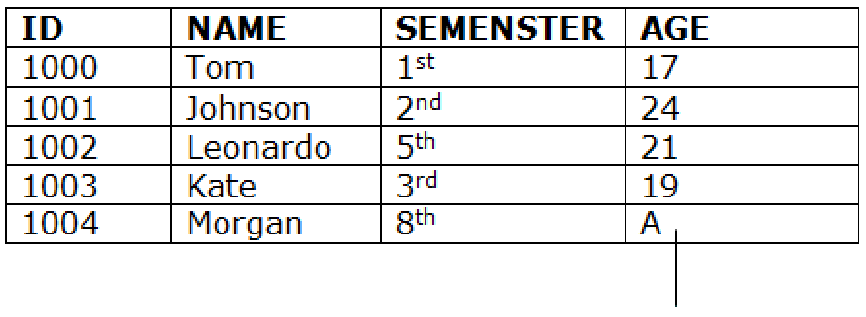
Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:



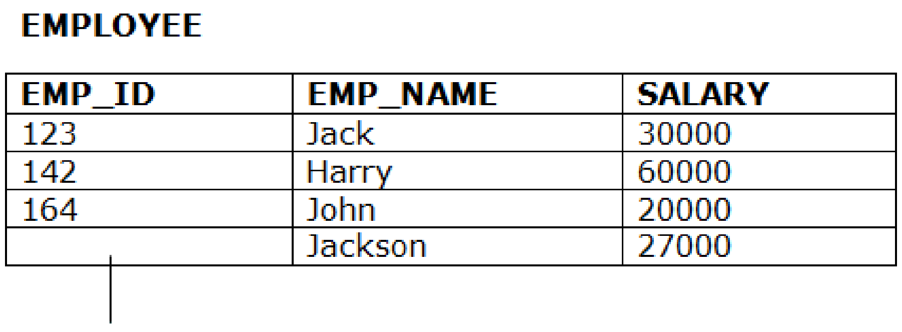
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:



EMPLOYEE

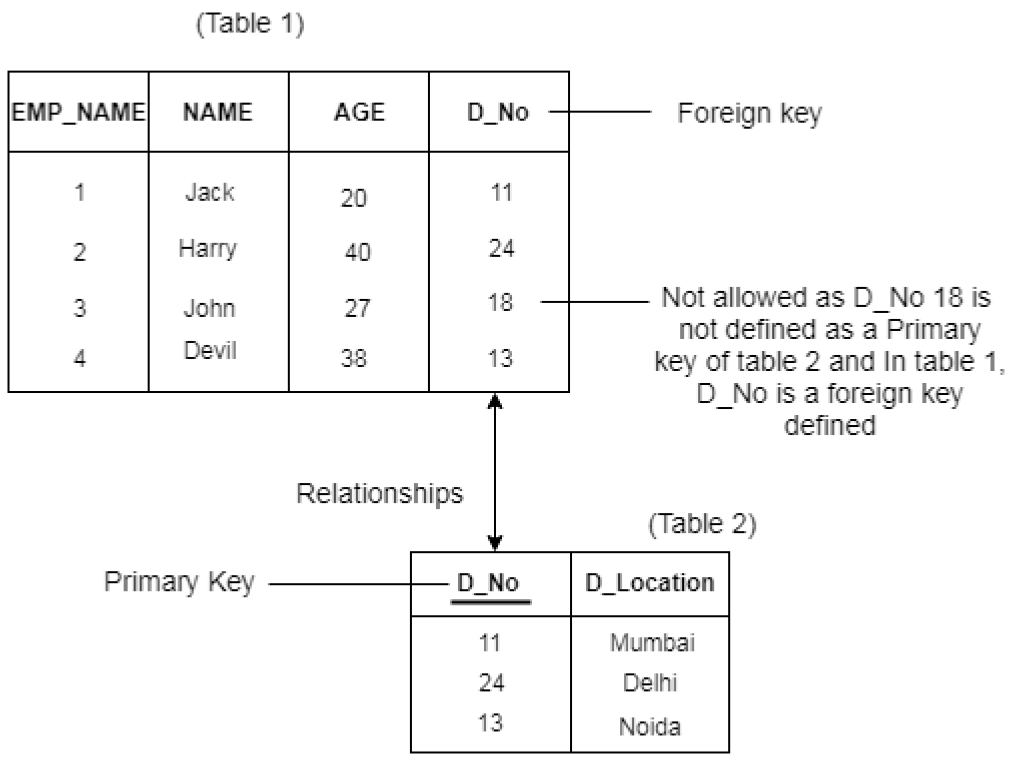
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:


(Table 1)
Foreign key
Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined
Relationships
(Table 2)
Primary Key

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

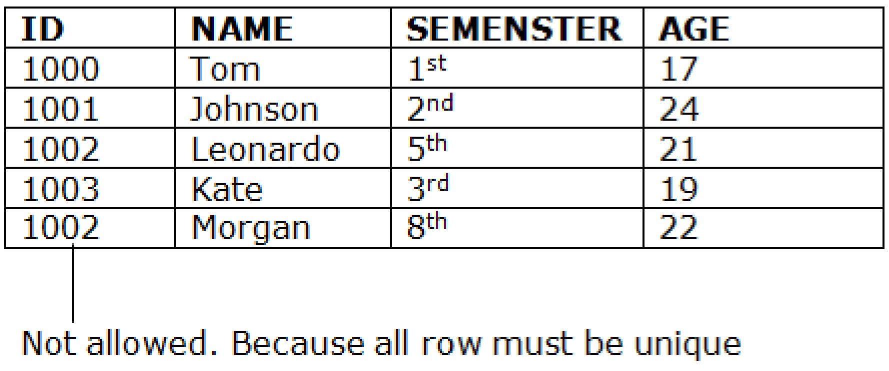
D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

Play Video



ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

• check constraints

Check Constraint is used to specify a predicate that every tuple must satisfy in a given relation. It limits the values that a column can hold in a relation.

- The predicate in check constraint can hold a sub query.
- Check constraint defined on an attribute restricts the range of values for that attribute.
- If the value being added to an attribute of a tuple violates the check constraint, the check constraint evaluates to false and the corresponding update is aborted.
- Check constraint is generally specified with the CREATE TABLE command in SQL.

Syntax:

```
CREATE TABLE pets(

        ID INT NOT NULL,

        Name VARCHAR(30) NOT NULL,

        Breed VARCHAR(20) NOT NULL,

        Age INT,

        GENDER VARCHAR(9),

        PRIMARY KEY(ID),

        check(GENDER in ('Male', 'Female', 'Unknown'))

        );
```

Note: The check constraint in the above SQL command restricts the GENDER to belong to only the categories specified. If a new tuple is added or an existing tuple in the relation is updated with a GENDER that doesn't belong to any of the three categories mentioned, then the corresponding database update is aborted.

Query

Constraint: Only students with age >= 17 are can enroll themselves in a university. **Schema for student database in university:**

```
CREATE TABLE student(

        StudentID INT NOT NULL,

        Name VARCHAR(30) NOT NULL,

        Age INT NOT NULL,

        GENDER VARCHAR(9),

        PRIMARY KEY(ID),

        check(Age >= 17)

        );
```

Student relation:

StudentID	Name	Age	Gender
1001	Ron	18	Male
1002	Sam	17	Male

1003	**Georgia**	**17**	**Female**
1004	**Erik**	**19**	**Unknown**
1005	**Christine**	**17**	**Female**

Explanation: In the above relation, the age of all students is greater than equal to 17 years, according to the constraint mentioned in the check statement in the schema of the relation. If, however following SQL statement is executed:

```
INSERT INTO student(STUDENTID, NAME, AGE, GENDER)

VALUES (1006, 'Emma', 16, 'Female');
```

There won't be any database update and as the age < 17 years. **Different options to use Check constraint:**

- **With alter:** Check constraint can also be added to an already created relation using the syntax:

```
alter table TABLE_NAME modify COLUMN_NAME check(Predicate);
```

- **Giving variable name to check constraint:**Check constraints can be given a variable name using the syntax:

```
alter table TABLE_NAME add constraint CHECK_CONST check (Predicate);
```

- **Remove check constraint:** Check constraint can be removed from the relation in the database from SQL server using the syntax:

```
alter table TABLE_NAME drop constraint CHECK_CONSTRAINT_NAME;
```

- **Drop check constraint:** Check constraint can be dropped from the relation in the database in MySQL using the syntax:

```
alter table TABLE_NAME drop check CHECK_CONSTRAINT_NAME;
```

View existing constraints on a particular table

If you want to check if a constraint or any constraint exists within the table in mysql then you can use the following command. This command will show a tabular output of all the constraint-related data for the table name you've passed in the statement, in our case we'll use the employee table.

```
SELECT *

FROM information_schema.table_constraints

WHERE table_schema = schema()

AND table_name = 'employee';
```

- **Data Manipulation commands**
- Data manipulation commands are used to manipulate data in the database.
- Some of the Data Manipulation Commands are-
- Select

- Select statement retrieves the data from database according to the constraints specifies alongside.
- SELECT <COLUMN NAME>
- FROM <TABLE NAME>
- WHERE <CONDITION>
- GROUP BY <COLUMN LIST>
- HAVING <CRITERIA FOR FUNCTION RESULTS>
- ORDER BY <COLUMN LIST>
- General syntax –
- Example: select * from employee where e_id>100;
- ## Insert
- Insert statement is used to insert data into database tables.
- General Syntax –
- INSERT INTO <TABLE NAME> (<COLUMNS TO INSERT>) VALUES (<VALUES TO INSERT>)
- Example: insert into Employee (name, dept_id) values ('ABC', 3);
- ## Update
- The update command updates existing data within a table.
- General syntax –
- UPDATE <TABLE NAME>
- SET <COLUMN NAME> = <UPDATED COLUMN VALUE>,
- <COLUMN NAME> = <UPDATED COLUMN VALUE>,
- <COLUMN NAME> = <UPDATED COLUMN VALUE>,…
- WHERE <CONDITION>
- Example: update Employee set Name='AMIT' where E_id=5;
- ## delete
- Deletes records from the database table according to the given constraints.
- General Syntax –
- DELETE FROM <TABLE NAME>
- WHERE <CONDITION>
- Example –
- delete from Employee where e_id=5;
- To delete all records from the table –

```
• Delete * from <TABLE NAME>;
```

- ## Merge
- Use the MERGE statement to select rows from one table for update or insertion into another table. The decision whether to update or insert into the target table is based on a condition in the ON clause. It is also known as UPSERT i.e. combination of UPDATE and INSERT.
- General Syntax (SQL) –

```
• MERGE <TARGET TABLE> [AS TARGET]
• USING <SOURCE TABLE> [AS SOURCE]
• ON <SEARCH CONDITION>
• [WHEN MATCHED
• THEN <MERGE MATCHED  > ]
• [WHEN NOT MATCHED [BY TARGET]
• THEN < MERGE NOT MATCHED >]
• [WHEN NOT MATCHED BY SOURCE
• THEN <MERGE MATCHED  >];
```

- General Syntax (Oracle)

```
• MERGE INTO <TARGET TABLE>
• USING <SOURCE TABLE>
• ON <SEARCH CONDITION>
• [WHEN MATCHED
• THEN <MERGE MATCHED > ]
• [WHEN NOT MATCHED
• THEN < MERGE NOT MATCHED > ];
```

- ## Data Control commands
- The Data Control Language (DCL) is a subset of the Structured Query Lanaguge (SQL) that allows database administrators to configure security access to relational databases.
- DCL is the simplest of the SQL subsets, as it consists of only three commands:

  i. **GRANTCommand-** Give user access privileges to a database

    - Oracle operates a closed system in that you cannot perform any action at all unless you have been authorized to do so. This includes logging onto the database, creating tables, views, indexes and synonyms, manipulating data in tables created by other users, etc.

- The SQL command to grant a privilege on a table: GRANT SELECT, INSERT, UPDATE, DELETE ON tablename TO username;
- Example:

  GRANT SELECT ON employee TO cr7; GRANT SELECT, UPDATE, DELETE ON employee TO cr7;

ii. **REVOKE Command-** withdraws access privileges given with the GRANT or taken with the DENY command

- The SQL command to revoke a privilege on a table: REVOKE SELECT, INSERT, UPDATE, DELETE ON tablename FROM username;
- Example:

  REVOKE SELECT ON employee FROM cr7;

  REVOKE SELECT, UPDATE, DELETE FROM cr7;

iii. **Deny Command-** deny user access

- The DENY command may be used to explicitly prevent a user from receiving a particular permission.
- This is helpful when a user may be a member of a role or group that is granted a permission and you want to prevent that user from inheriting the permission by creating an exception.
- The SQL Command for this:

  DENY [permission]

  ON [object]

  TO [user]

  Example:

  DENY DELETE

  ON HR.employees

- Access is a four-part relationship -

- **Grantor-** a user or admin who controls the privileges on a Schema Object
- **Privileges-** the actions that can be done on a Schema Object
- **Schema Object-** Table, View, Domain, Collation, stored procedure, trigger, etc.
- **Grantee-** a user who is given Privileges on aSchema Object

- ## Set and string operations

- **Set Operations**

Refer assignment 2

String functions are used to perform an operation on input string and return an output string.
Following are the string functions defined in SQL:

1. **ASCII():** This function is used to find the ASCII value of a character.

```
Syntax: SELECT ascii('t');
Output: 116
```

2. **CHAR_LENGTH():** Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a word.

```
Syntax: SELECT char_length('Hello!');
Output: 6
```

3. **CHARACTER_LENGTH():** Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a line.

```
Syntax: SELECT CHARACTER_LENGTH('geeks for geeks');
Output: 15
```

4. **CONCAT():** This function is used to add two words or strings.

```
Syntax: SELECT 'Geeks' || ' ' || 'forGeeks' FROM dual;
Output: 'GeeksforGeeks'
```

5. **CONCAT_WS():** This function is used to add two words or strings with a symbol as concatenating symbol.

```
Syntax: SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');
Output: geeks_for_geeks
```

6. **FIND_IN_SET():** This function is used to find a symbol from a set of symbols.

```
Syntax: SELECT FIND_IN_SET('b', 'a, b, c, d, e, f');
Output: 2
```

7. **FORMAT():** This function is used to display a number in the given format.

```
Syntax: Format("0.981", "Percent");
Output: '98.10%'
```

8. **INSERT():** This function is used to insert the data into a database.

```
Syntax: INSERT INTO database (geek_id, geek_name) VALUES (5000, 'abc');
Output: successfully updated
```

9. **INSTR():** This function is used to find the occurrence of an alphabet.

```
Syntax: INSTR('geeks for geeks', 'e');
Output: 2 (the first occurrence of 'e')
```

```
Syntax: INSTR('geeks for geeks', 'e', 1, 2 );
Output: 3 (the second occurrence of 'e')
```

10. **LCASE():** This function is used to convert the given string into lower case.

```
Syntax: LCASE ("GeeksFor Geeks To Learn");
Output: geeksforgeeks to learn
```

11. **LEFT():** This function is used to SELECT a sub string from the left of given size or characters.

```
Syntax: SELECT LEFT('geeksforgeeks.org', 5);
Output: geeks
```

12. **LENGTH():** This function is used to find the length of a word.

```
Syntax: LENGTH('GeeksForGeeks');
Output: 13
```

13. **LOCATE():** This function is used to find the nth position of the given word in a string.

```
Syntax: SELECT LOCATE('for', 'geeksforgeeks', 1);
Output: 6
```

14. **LOWER():** This function is used to convert the upper case string into lower case.

```
Syntax: SELECT LOWER('GEEKSFORGEEKS.ORG');
Output: geeksforgeeks.org
```

15. **LPAD():** This function is used to make the given string of the given size by adding the given symbol.

```
Syntax: LPAD('geeks', 8, '0');
Output:
000geeks
```

16. **LTRIM():** This function is used to cut the given sub string from the original string.

```
Syntax: LTRIM('123123geeks', '123');
Output: geeks
```

17. **MID():** This function is to find a word from the given position and of the given size.

```
Syntax: Mid ("geeksforgeeks", 6, 2);
Output: for
```

18. **POSITION():** This function is used to find position of the first occurrence of the given alphabet.

```
Syntax: SELECT POSITION('e' IN 'geeksforgeeks');
Output: 2
```

19. **REPEAT():** This function is used to write the given string again and again till the number of times mentioned.

```
Syntax: SELECT REPEAT('geeks', 2);
Output: geeksgeeks
```

20. **REPLACE():** This function is used to cut the given string by removing the given sub string.

```
Syntax: REPLACE('123geeks123', '123');
Output: geeks
```

21. **REVERSE():** This function is used to reverse a string.

```
Syntax: SELECT REVERSE('geeksforgeeks.org');
Output: ‘gro.skeegrofskeeg’
```

22. **RIGHT():** This function is used to SELECT a sub string from the right end of the given size.

```
Syntax: SELECT RIGHT('geeksforgeeks.org', 4);
Output: ‘.org’
```

23. **RPAD():** This function is used to make the given string as long as the given size by adding the given symbol on the right.

```
Syntax: RPAD('geeks', 8, '0');
Output: ‘geeks000’
```

24. **RTRIM():** This function is used to cut the given sub string from the original string.

```
Syntax: RTRIM('geeksxyxzyyy', 'xyz');
Output: ‘geeks’
```

25. **SPACE():** This function is used to write the given number of spaces.

```
Syntax: SELECT SPACE(7);
Output: ‘         ‘
```

26. **STRCMP():** This function is used to compare 2 strings.
    - If string1 and string2 are the same, the STRCMP function will return 0.
    - If string1 is smaller than string2, the STRCMP function will return -1.
    - If string1 is larger than string2, the STRCMP function will return 1.

```
Syntax: SELECT STRCMP('google.com', 'geeksforgeeks.com');
Output: -1
```

27. **SUBSTR():** This function is used to find a sub string from the a string from the given position.

```
Syntax:SUBSTR('geeksforgeeks', 1, 5);
Output: ‘geeks’
```

28. **SUBSTRING():** This function is used to find an alphabet from the mentioned size and the given string.

```
Syntax: SELECT SUBSTRING('GeeksForGeeks.org', 9, 1);
Output: ‘G’
```

29. **SUBSTRING_INDEX():** This function is used to find a sub string before the given symbol.

```
Syntax: SELECT SUBSTRING_INDEX('www.geeksforgeeks.org', '.', 1);
Output: ‘www’
```

30. **TRIM():** This function is used to cut the given symbol from the string.

```
Syntax: TRIM(LEADING '0' FROM '000123');
Output: 123
```

31. **UCASE():** This function is used to make the string in upper case.

```
Syntax: UCASE ("GeeksForGeeks");
Output:
GEEKSFORGEEKS
```

- **aggregate function-group by, having**

Q. How to Use HAVING With Aggregate Functions in SQL?

Q. When to use the **HAVING** keyword?

SQL provides many built-in functions to perform tasks. There are 4 types of functions Date functions, Char functions, Numeric functions, and Aggregate functions.

Here we will be looking at aggregate functions and how to use them with the **HAVING** keyword.
Aggregate Functions are database built-in functions that act on multiple rows of the table and produce a single output. There are basically 5 aggregate functions that we use frequently in SQL. Aggregate functions are deterministic. Common aggregate functions are as follows:

- **COUNT():** Calculates the total number of rows in the table, it returns a single value.
- **AVG():** Calculates the average of the values to the column it is applied to.
- **MIN():** Returns the minimum value in the column it is applied to.
- **MAX():** Returns the maximum value in the column it is applied to.
- **SUM():** Return the sum of all values of the column it is applied to.

WHERE keyword that we used to filter data on the given condition works well with SQL operators like arithmetic operator, comparison operator, etc but when it comes to aggregate functions we use the HAVING keyword to sort data on the given condition. The GROUP BY clause is also used with the HAVING keyword.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY expression
HAVING condition
ORDER BY expression
LIMIT value;
```

To use SUM() with Having clause:
Step 1: Create a database
Query:

```
CREATE DATABASE database_name;
```

Step 2: Create a table named products.

Query:

```
CREATE TABLE PRODUCTS(product_id int primary key, product_name varchar(45),
product_cost float);
```

Step 3: Insert values in the table

Query:

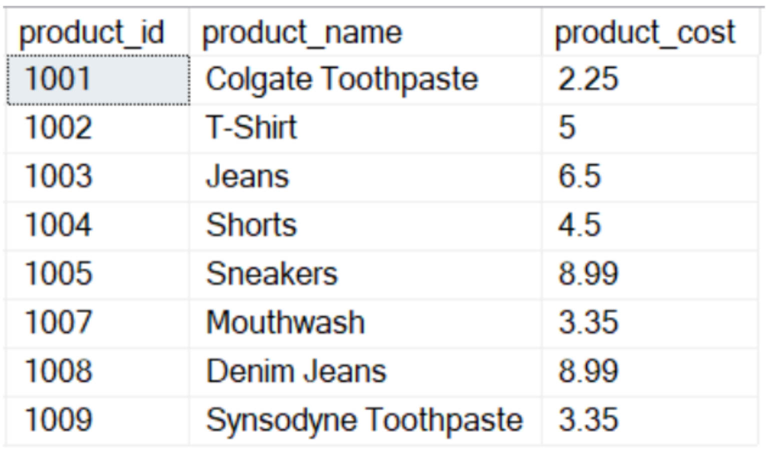
```
INSERT INTO PRODUCTS VALUES
(1001, 'Colgate Toothpaste', 2.25), (1002 'T-Shirt', 5),
(1003, 'Jeans', 6.5), (1004, 'Shorts', 4.5),
(1005, 'Sneakers', 8.99), (1007, 'Mouthwash', 3.35),
(1008, 'Denim Jeans', 8.99), (1009, 'Synsodyne Toothpaste', 3.35);
```

Step 4: Now let's see the contents of the products table.

Query:

```
SELECT * FROM products;
```

Output:



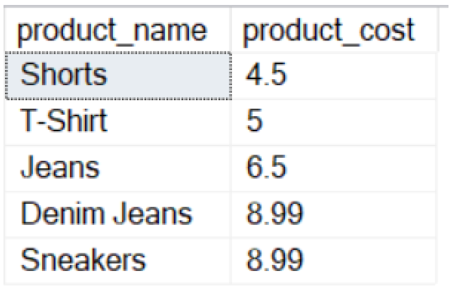
product_id	product_name	product_cost
1001	Colgate Toothpaste	2.25
1002	T-Shirt	5
1003	Jeans	6.5
1004	Shorts	4.5
1005	Sneakers	8.99
1007	Mouthwash	3.35
1008	Denim Jeans	8.99
1009	Synsodyne Toothpaste	3.35

Step 5: Now our task is to print all those products whose sum of product cost is greater than 3.50.

Query:

```
SELECT product_name, product_cost
FROM products
GROUP BY product_name, product_cost
HAVING SUM(product_cost) > 3.5
ORDER BY product_cost;
```

Output:



product_name	product_cost
Shorts	4.5
T-Shirt	5
Jeans	6.5
Denim Jeans	8.99
Sneakers	8.99

PRODUCTS TABLE

Here only those products are displayed whose cost is greater than 3.5

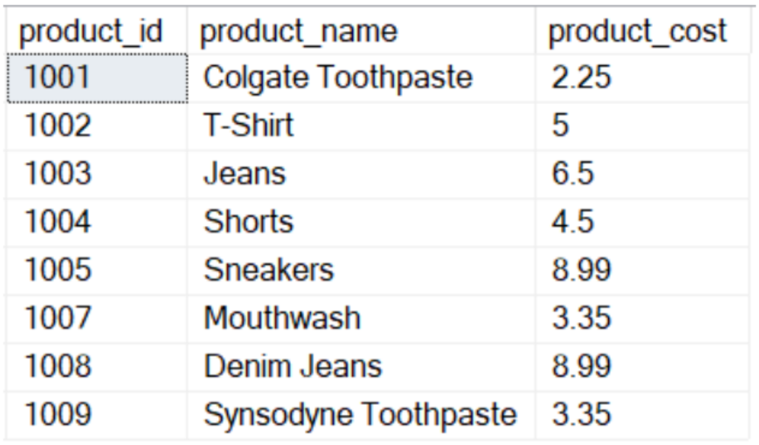
To use MAX() and MIN() with Having clause

We are using the same products table that we used in the previous example.

Our task is to find the products name whose maximum price is greater than 7 and those products names whose minimum price is less than 3.

Query:

```
SELECT * FROM products;
```

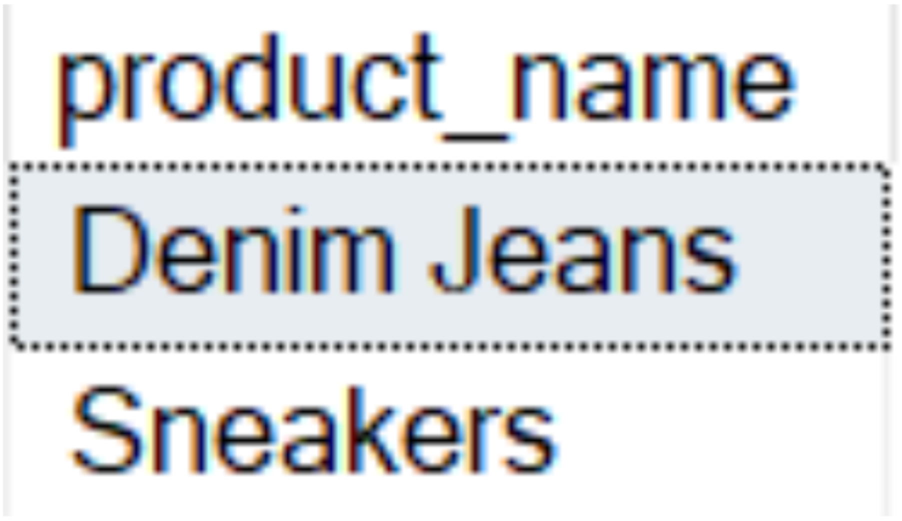


product_id	product_name	product_cost
1001	Colgate Toothpaste	2.25
1002	T-Shirt	5
1003	Jeans	6.5
1004	Shorts	4.5
1005	Sneakers	8.99
1007	Mouthwash	3.35
1008	Denim Jeans	8.99
1009	Synsodyne Toothpaste	3.35

QUERY 1 (To find products with a maximum price greater than 7)

```
SELECT product_name
FROM products
GROUP BY product_name
HAVING MAX(product_cost) > 7;
```

OUTPUT

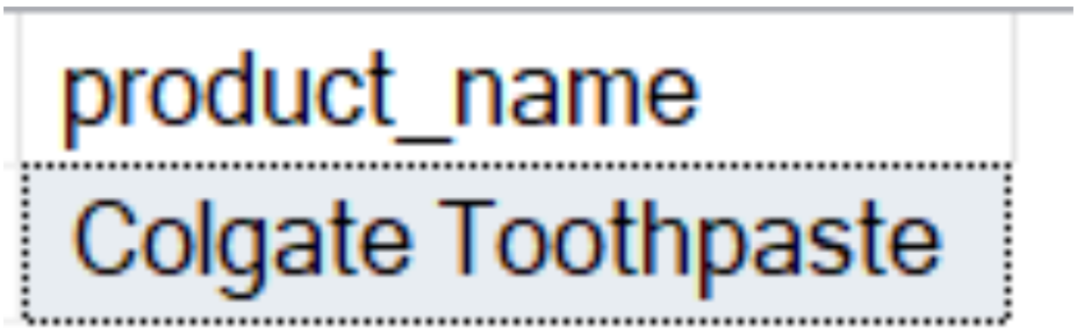


product_name
Denim Jeans
Sneakers

QUERY 2(To find products with a minimum price less than 3)

```
SELECT product_name
FROM products
GROUP BY product_name
HAVING MIN(product_cost) < 3;
```

Output:



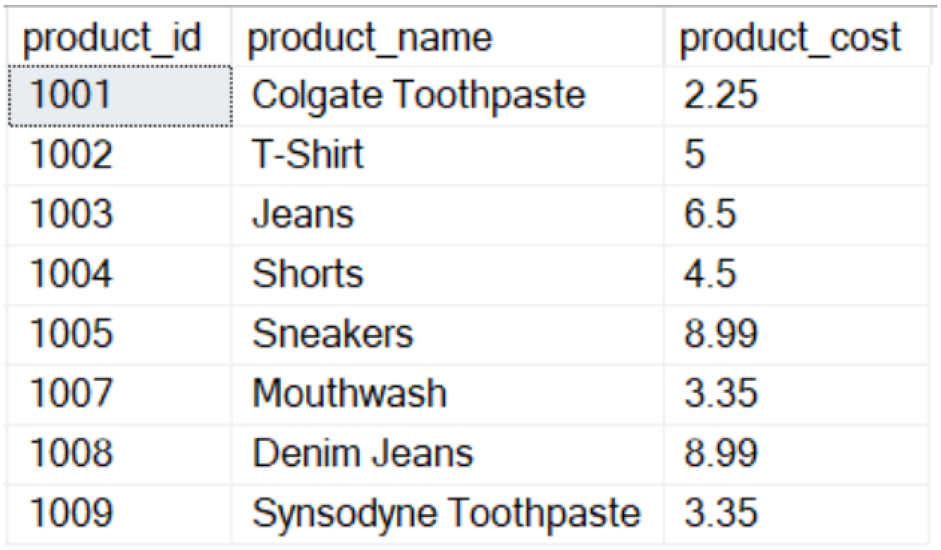
```
product_name
Colgate Toothpaste
```

To use AVG() with Having clause

We will be using the products table to demonstrate this part.

Query:

```
SELECT * FROM products;
```



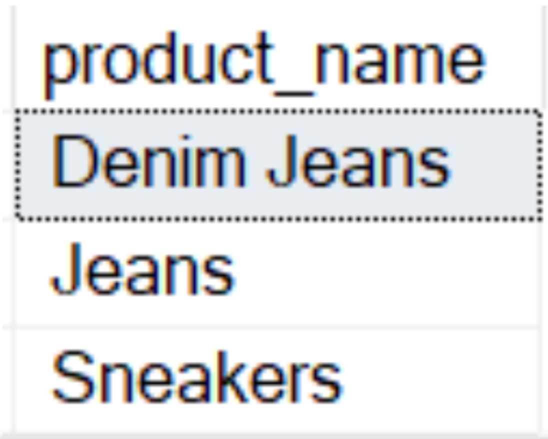
product_id	product_name	product_cost
1001	Colgate Toothpaste	2.25
1002	T-Shirt	5
1003	Jeans	6.5
1004	Shorts	4.5
1005	Sneakers	8.99
1007	Mouthwash	3.35
1008	Denim Jeans	8.99
1009	Synsodyne Toothpaste	3.35

Now, we want to select those products whose price is greater than the average price of the products table.

Query:

```
SELECT product_name
FROM products
GROUP BY product_name
HAVING AVG(product_cost) > (SELECT AVG(product_cost) FROM products);
```

Output:



product_name
Denim Jeans
Jeans
Sneakers

Here only those products are present whose average price is greater than the average price of the products table.

To use Count() with Having clause

Step 1: We will create a database.

Query:

```
CREATE DATABASE database_name;
```

Step 2: Create table login.

Query:

```
CREATE TABLE login(signin_id int PRIMARY KEY ,customer_id int, date_login
date);
```

Step 3: Insert values in the table.

Query:

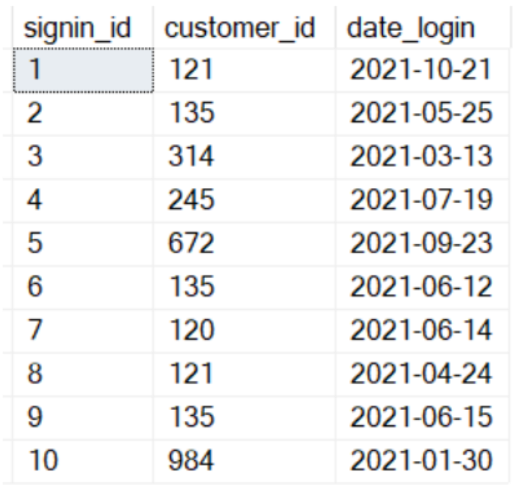
```
INSERT INTO login values
(1, 121, '2021-10-21'), (2, 135, '2021-05-25'),
(3, 314, '2021-03-13'), (4, 245, '2021-07-19'),
(5, 672, '2021-09-23'), (6, 135, '2021-06-12'),
(7,120,'2021-06-14'), (8, 121, '2021-04-24'),
(9,135, '2021-06-15'), (10, 984, '2021-01-30');
```

Step 4: Display the content of the table.

Query:

```
SELECT * FROM login;
```

Output:



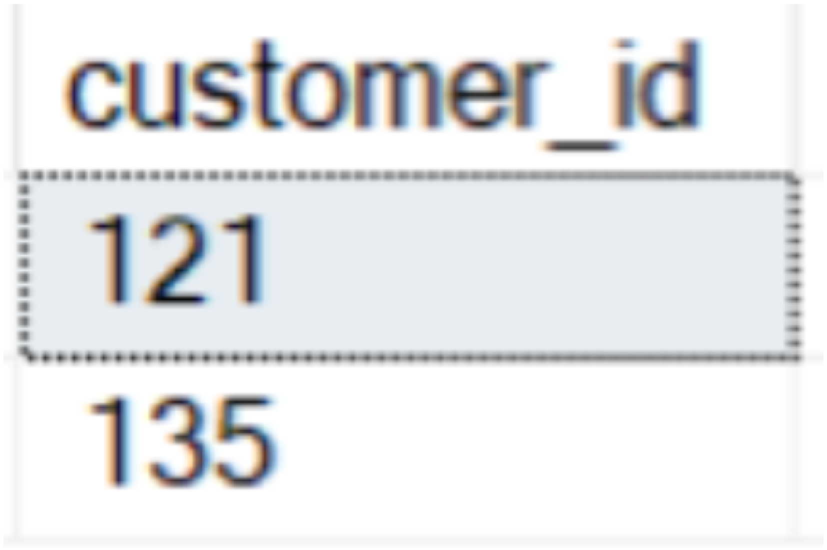
signin_id	customer_id	date_login
1	121	2021-10-21
2	135	2021-05-25
3	314	2021-03-13
4	245	2021-07-19
5	672	2021-09-23
6	135	2021-06-12
7	120	2021-06-14
8	121	2021-04-24
9	135	2021-06-15
10	984	2021-01-30

Now we want to display those customer ids (s) that occurred at least 2 times.

Query:

```
SELECT customer_id
FROM login
 GROUP BY customer_id
HAVING COUNT(customer_id) >=2 ;
```

Output:



customer_id
121
135

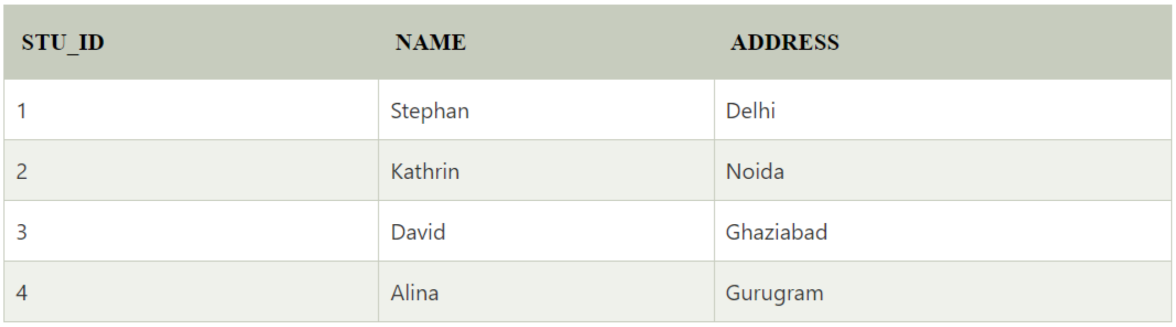
Here customer_id 121 and 135 occurred at least 2 times.

- **Views in SQL**

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

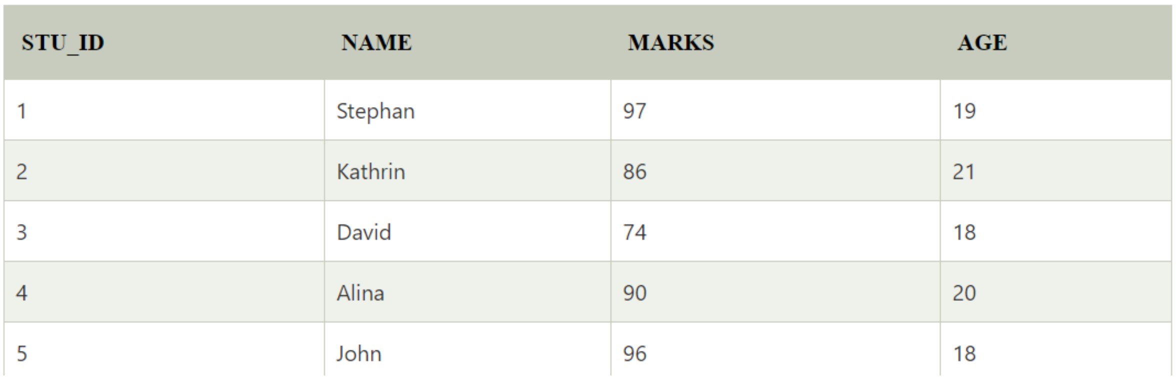
Sample table:

Student_Detail



STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks



STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

3. Creating View from multiple tables

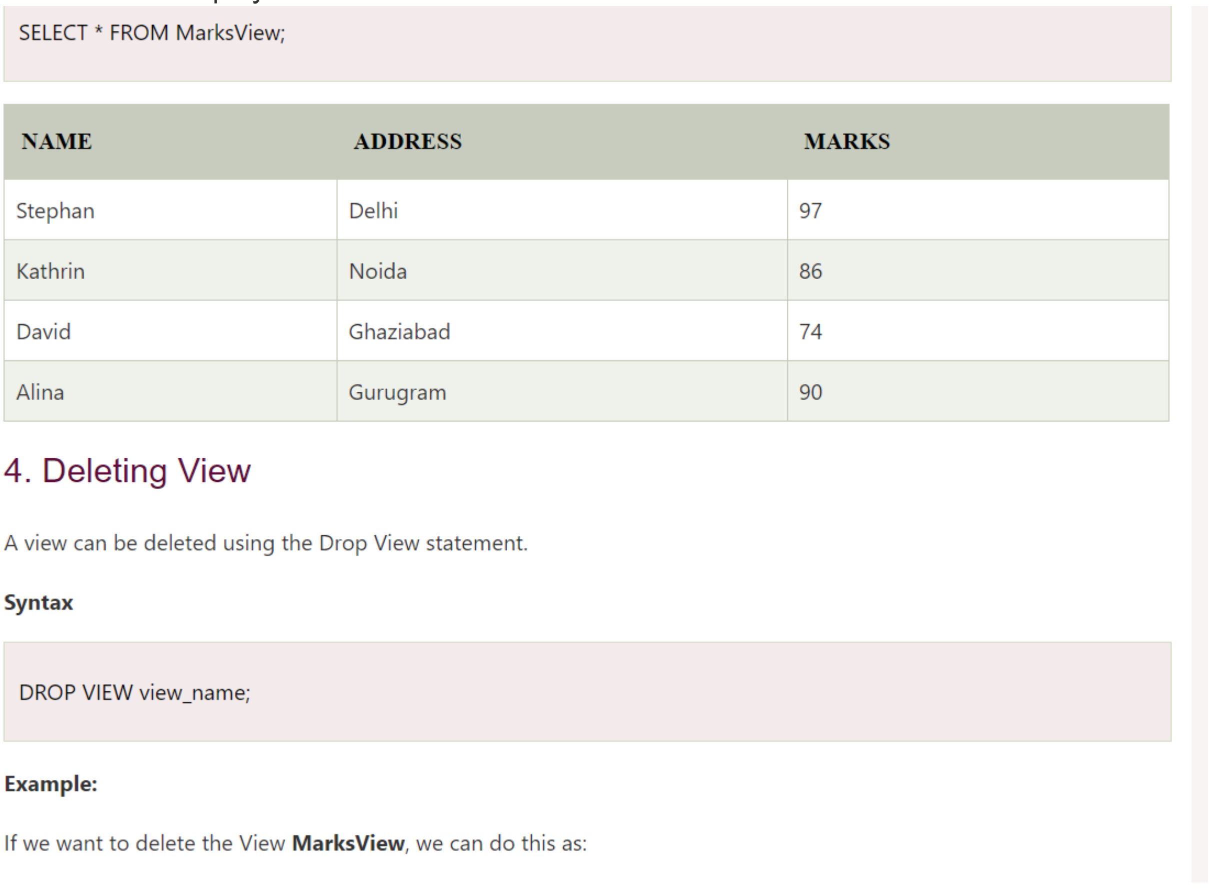
View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:



```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

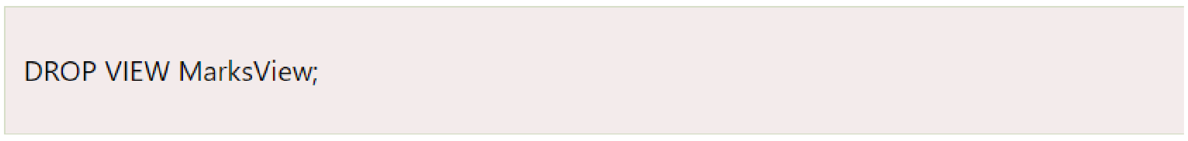
A view can be deleted using the Drop View statement.

Syntax

```
DROP VIEW view_name;
```

Example:

If we want to delete the View **MarksView**, we can do this as:



```
DROP VIEW MarksView;
```

- **joins, Nested and complex queries**

The growth of technology and automation coupled with exponential amounts of data has led to the importance and omnipresence of databases which, simply put, are organized collections of data. Considering a naive approach, one can theoretically keep all the data in one large table, however that increases the access time in searching for a record, security issues if the master table is destroyed, redundant storage of information and other issues. So tables are decomposed into multiple smaller tables.

For retrieving information from multiple tables, we need to extract selected data from different records, using operations called join(inner join, outer join and most importantly natural join). Consider 2 table schemas employee(employee_name, street, city)with n rows and works(employee_name, branch_name, salary) with m rows. A cartesian product of these 2 tables creates a table with n*m rows. A natural join selects from this n*m rows all rows with same values for employee_name. To avoid loss of information(some tuples in employee have no corresponding tuples in works) we use left outer join or right outer join.

A join operation or a nested query is better subject to conditions:

- Suppose our 2 tables are stored on a local system. Performing a join or a nested query will make little difference. Now let tables be stored across a distributed databases. For a nested query, we only extract the relevant information from each table, located on different computers, then merge the tuples obtained to obtain the result. For a join, we would be required to fetch the whole table from each site and create a large table from which the filtering will occur, hence more time will be required. So for distributed databases, nested queries are better.
- RDBMS optimizer is concerned with performance related to the subquery or join written by the programmer. Joins are universally understood hence no optimization issues can arise. If portability across multiple platforms is called for, avoid subqueries as it may run into bugs(SQL server more adept with joins as its usually used with Microsoft's graphical query editors that use joins).
- Implementation specific: Suppose we have queries where a few of the nested queries are constant. In MySQL, every constant subquery would be evaluated as many times as encountered, there being no cache facility. This is an obvious problem if the constant subquery involves large tuples. Subqueries return a set of data. Joins return a dataset which is necessarily indexed. Working on indexed data is faster so if the dataset returned by subqueries is large, joins are a better idea.
- Subqueries may take longer to execute than joins depending on how the database optimizer treats them(may be converted to joins). Subqueries are easier to read, understand and evaluate than cryptic joins. They allow a bottom-up approach, isolating and completing each task sequentially.

Join operation and nested queries are both used in relational database management systems (RDBMS) to combine data from multiple tables, but they differ in their approach.

Join operation:
A join operation combines data from two or more tables based on a common column or columns. The join operation is performed using the JOIN keyword in SQL, and it returns a single result set that contains columns from all the tables involved in the join.
For example, let's say we have two tables, Table1 and Table2, with the following data:

Table1
ID | Name
1 | John
2 | Sarah

3	David

Table2

ID	Address
1	123 Main St.
2	456 Elm St.
4	789 Oak St.

If we want to combine the data from these two tables based on the ID column, we can perform an inner join using the following SQL query:

```
SELECT Table1.ID, Table1.Name, Table2.Address
FROM Table1
INNER JOIN Table2
ON Table1.ID = Table2.ID
```

Result:

```
ID | Name | Address
1 | John | 123 Main St.
2 | Sarah | 456 Elm St.
```

Nested query:
A nested query is a SQL query that is embedded within another SQL query. It is used to retrieve data from one or more tables based on a condition that is evaluated using the results of another query. The nested query is executed first, and its results are used to evaluate the outer query.
For example, let’s say we have the same two tables as before:

Table1

ID	Name
1	John
2	Sarah
3	David

Table2

ID	Address
1	123 Main St.
2	456 Elm St.
4	789 Oak St.

If we want to retrieve the names of the people who have an address in Table2, we can use a nested query as follows:

```
SELECT Name
FROM Table1
WHERE ID IN (SELECT ID FROM Table2)
```

Result:

```
Name
John
Sarah
```

In this case, the nested query is executed first, and it returns the ID values of the rows in Table2. These ID values are then used to evaluate the outer query, which retrieves the names of the people who have those ID values in Table1.

The choice between using a join operation or a nested query depends on the specific requirements of the task at hand. Joins are often faster and more efficient for large datasets, but nested queries can be more flexible and allow for more complex conditions to be evaluated.

- ## Triggers (ECA Model)

I. Introduction:

1. A trigger is a special kind of a store procedure that executes in response to certain action on the table like insertion, deletion or updating of data.
2. In many cases it is convenient to specify the type of action to be taken when certain events occur and when certain conditions are satisfied.
3. Consider the following condition where the sales manager needs to receive a mail whenever they receive an order of high propriety. In this case, a trigger can be written to determine priority of the order received and send mail to sales head.

II. ECA Model:

Trigger is based on the model named ECA i.e. Event-Condition-Action.

1. Event:

a) The rules are triggered by Event.

b) These events can be database insert, update or delete statements.

c) Changing an employee’s manager can be an event on which trigger can be fired.

2. Condition:

a) The condition that determines whether the rule action should be executed.

b) Once the triggering event has occurred, an optional condition may be evaluated.

c) If no condition is specified, the action will be executed once the event occurs.

d) If a condition is specified, it is first evaluated, and only if it evaluates to true will the rule action be executed.

3. Action: a) The action is usually a sequence of SQL statements.

b) It could also be a database transaction or an external program that will be automatically executed.

c) For example, executing a stored procedure when a particular event occurs.

III. Example:

Here we are using the syntax of SQL Server database system.

Consider the following table Tbl_Employee with columns Emp_ID, Emp_Name and Emp_Sal.

Emp_ID	**Emp_Name**	**Emp_Sal**
1	ABC	1000.00
2	DEF	1200.00
3	GHI	1100.00
4	JKL	1300.00
5	MNO	1400.00

Consider the following requirement in which we need to capture any changes made to Tbl_Employee shown above into another Audit Table named Tbl_EmpAudit as shown below.

| Emp_ID | Emp_Name | Emp_Sal | Audit_Action | Audit_Timestamp | |--------|----------|---------|--------------|-----------------|

A trigger needs to be fired when a new row is inserted into Tbl_Employee. The trigger makes an entry into the Tbl_EmpAudit. Following is the trigger which gets trigger which will be fired when a new row is inserted into

Tbl_Employee.

CREATE TRIGGER Trg_AfterInsert ON [Tbl_Employee]

FOR INSERT

AS DECLARE @empid INT; DECLARE @empname VARCHAR(100); DECLARE @empsal DECIMAL(10,2); DECLARE @audit_action VARCHAR(100);

```
SELECT @empid=i.Emp_ID FROM inserted i;


    SELECT @empname=i.Emp_Name FROM inserted i;
    SELECT @empsal=i.Emp_Sal FROM inserted i;
SET @audit_action='Inserted Record -- After Insert Trigger.';


INSERT INTO Tbl_EmpAudit
 (
 Emp_ID,
 Emp_Name,
 Emp_Sal,
 Audit_Action,
 Audit_Timestamp
 )
VALUES
(
@empid,
@empname,
@empsal,
@audit_action,
getdate()
);


PRINT 'AFTER INSERT trigger fired.'
```

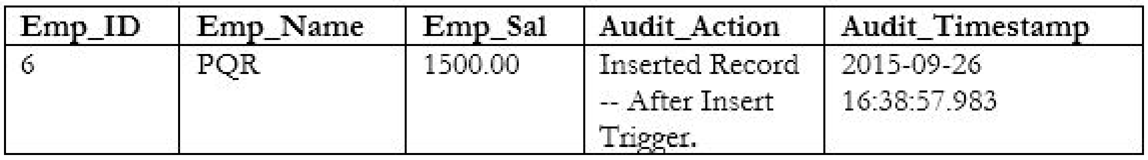
GO

To invoke the trigger, row needs to be inserted to Tbl_Employee. Consider the following insert statement.

INSERT INTO Tbl_Employee VALUES('PQR',1500);

The trigger is fired and an entry is made in the table Tbl_EmpAudit.

The entry in table Tbl_EmpAudit is as follows:



Emp_ID	Emp_Name	Emp_Sal	Audit_Action	Audit_Timestamp
6	PQR	1500.00	Inserted Record -- After Insert Trigger.	2015-09-26 16:38:57.983

- **Security and Authorization in SQL**

Database security has many different layers, but the key aspects are:

Authentication

User authentication is to make sure that the person accessing the database is who he claims to be. Authentication can be done at the operating system level or even the database level itself. Many authentication systems such as retina scanners or bio-metrics are used to make sure unauthorized people cannot access the database.

Authorization

Authorization is a privilege provided by the Database Administer. Users of the database can only view the contents they are authorized to view. The rest of the database is out of bounds to them.

The different permissions for authorizations available are:

- **Primary Permission -** This is granted to users publicly and directly.
- **Secondary Permission -** This is granted to groups and automatically awarded to a user if he is a member of the group.
- **Public Permission -** This is publicly granted to all the users.
- **Context sensitive permission -** This is related to sensitive content and only granted to a select users.

The categories of authorization that can be given to users are:

- **System Administrator -** This is the highest administrative authorization for a user. Users with this authorization can also execute some database administrator commands such as restore or upgrade a database.
- **System Control -** This is the highest control authorization for a user. This allows maintenance operations on the database but not direct access to data.
- **System Maintenance -** This is the lower level of system control authority. It also allows users to maintain the database but within a database manager instance.
- **System Monitor -** Using this authority, the user can monitor the database and take snapshots of it.