UNIT -3

Queries, Constraints, Triggers

SYLLABUS

UNIT-III

- The Form of Basic SQL Query
- Union
- Intersect and Except
- Nested Queries
- Aggregate Operators
- Null Values
- Complex Integrity Constraints in SQL
- Triggers and Active Database.

The Form of Basic SQL Query:-

Introduction to SQL:-

SQL is a database computer language designed for the retrieval and management of data in a relational database. SQL stands for Structured Query Language. This tutorial will give you a quick start to SQL. It covers most of the topics required for a basic understanding of SQL and to get a feel of how it works.

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an**A SI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

What is SQL?

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

SQL is the standard language for Relational Database System. 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.

Also, they are using different dialects, such as –

- MS SQL Server using T - SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

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.

A Brief History of SQL

- **1970** – Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974** – Structured Query Language appeared.
- **1978** – IBM worked to develop Codd's ideas and released a product named System/R.
- **1986** – IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

SQL Process

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.

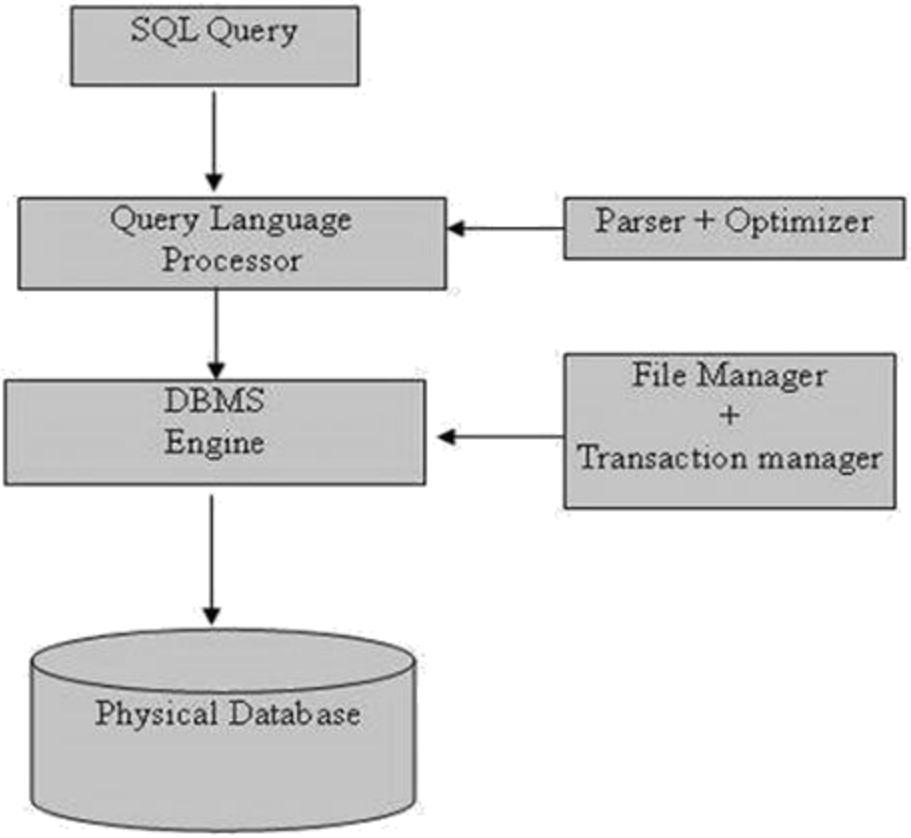
There are various components included in this process.

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 –



SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature –

DDL - Data Definition Language

Sr.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.

DML - Data Manipulation Language

Sr.No.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.
2	**INSERT** Creates a record.
3	**UPDATE** Modifies records.
4	**DELETE** Deletes records.

DCL - Data Control Language

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

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have

- The same number of columns selected
- The same number of column expressions
- The same data type and
- Have them in the same order

But they need not have to be in the same length.

Syntax

The basic syntax of a **UNION** clause is as follows –

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 – CUSTOMERS Table is as follows.

```
+----  +----------+-----  +-----------+.....          +
| ID | NAME      | AGE | ADDRESS    | SALARY     |
+----  +----------+-----  +-----------+.....          +
|   1 | Ramesh   |   32 | Ahmedabad |   2000.00  |
|   2 | Khilan   |   25 | Delhi     |   1500.00  |
|   3 | kaushik  |   23 | Kota      |   2000.00  |
|   4 | Chaitali |   25 | Mumbai    |   6500.00  |
|   5 | Hardik   |   27 | Bhopal    |   8500.00  |
```

```
|   6 | Komal    |   22 | MP         |   4500.00 |
|   7 | Muffy    |   24 | Indore     | 10000.00  |
+---- +----------+----- +------------+---------- +
```

Table 2 – ORDERS Table is as follows.

```
+-----                    +------------  +.....      +
|OID  | DATE                | CUSTOMER ID  | AMOUNT |
+-----                    +------------  +.....      +
|102|   2009-10-08 00:00:00 |            3 |    3000  |
|100|   2009-10-08 00:00:00 |            3 |    1500  |
|101|   2009-11-20 00:00:00 |            2 |    1560  |
|103|   2008-05-20 00:00:00 |            4 |    2060  |
+-----                    +------------  +.....      +
```

Now, let us join these two tables in our SE ECT statement as follows –

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   LEFT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION
   SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   RIG T JOIN ORDERS
   ON CU TOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

```
+------  +----------+.....      +--------------------- +
| ID   | NAME      | AMOUNT | DATE                 |
+------ +----------+-------- +--------------------- +
|    1 | Ramesh   |    NULL | NULL                 |
|    2 | Khilan   |    1560  | 2009-11-20 00:00:00 |
|    3 | kaushik  |    3000  | 2009-10-08 00:00:00 |
|    3 | kaushik  |    1500  | 2009-10-08 00:00:00 |
|    4 | Chaitali |    2060  | 2008-05-20 00:00:00 |
```

```
|     5| Hardik    |   NULL | NULL                 |
|     6| Komal     |   NULL | NULL                 |
|     7| Muffy     |   NULL | NULL                 |
+------+----------+.....     +--------------------- +
```

The UNION ALL Clause

The UNION ALL operator is used to combine the results of two ELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the U ION ALL operator.

Syntax

The basic syntax of the **UNION ALL** is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables,

Table 1 – CUSTOMERS Table is as follows.

```
+----  +----------+-----  +-----------+.....          +
|  D | NAME      | AGE | ADDRESS    | SALARY     |
+----  +----------+-----  +-----------+.....          +
|  1 | Ramesh    |   32 | Ahmedabad |   2000.00 |
|  2 | Khilan    |   25 | Delhi     |   1500.00 |
|  3 | kaushik   |   23 | Kota      |   2000.00 |
|  4 | Chaitali  |   25 | Mumbai    |   6500.00 |
|  5 | Hardik    |   27 | Bhopal    |   8500.00 |
|  6 | Komal     |   22 | MP        |   4500.00 |
```

```
|  7 | Muffy    |   24 | Indore     | 10000.00 |
+----+----------+------+------------+----------+
```

Table 2 – ORDERS table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER ID | AMOUNT |
+-----+---------------------+-------------+--------+
|102|   2009-10-08 00:00:00 |           3|    3000 |
| 100 | 2009-10-08 00:00:00 |           3|    1500 |
| 101 | 2009-11-20 00:00:00 |           2|    1560 |
| 103 | 2008-05-20 00:00:00 |           4|    2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SE ECT statement as follows –

```
SQL> SELECT ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   LEFT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CU TOMER_ID
UNION ALL
   SELECT ID, NAME,  MOUNT, DATE
   FROM CUSTOMERS
   RIG T JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

```
+------  +----------+--------     +--------------------- +
| ID   | NAME      | AMOUNT | DATE                  |
+------  +----------+--------     +--------------------- +
|     1 | Ramesh   |    NULL | NULL                  |
|     2 | Khilan   |    1560  | 2009-11-20 00:00:00 |
|     3 | kaushik  |    3000  | 2009-10-08 00:00:00 |
|     3 | kaushik  |    1500  | 2009-10-08 00:00:00 |
|     4 | Chaitali |    2060  | 2008-05-20 00:00:00 |
|     5 | Hardik   |    NULL | NULL                  |
```

```
|     6| Komal    |   NULL | NULL                |
|     7| Muffy    |   NULL | NULL                |
|     3| kaushik  |   3000 |  2009-10-08 00:00:00 |
|     3| kaushik  |   1500 |  2009-10-08 00:00:00 |
|     2| Khilan   |   1560 | 2009-11-20  00:00:00 |
|     4| Chaitali |   2060 | 2008-05-20  00:00:00 |
+------+----------+--------+·····                 +
```

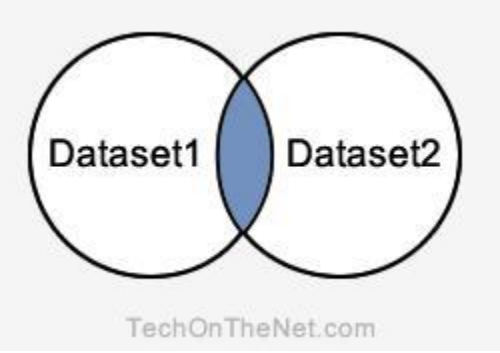
There are two other clauses (i.e., operators), which are like the UNION clause.

- ☐ SQL INTERSECT Clause – This is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.
- ☐ SQL EXCEPT Clause – This combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

INTERSECT OPERATION:-

Description

The SQL INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.



Explanation: The INTERSECT query will return the records in the blue shaded area. These are the records that exist in both Dataset1 and Dataset2.

Each SQL statement within the SQL INTERSECT must have the same number of fields in the result sets with similar data types.

Syntax

The syntax for the INTERSECT operator in SQL is:

```
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
INTERSECT
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];
```

Parameters or Arguments

expression1, expression2, expression_n

The columns or calculations that you wish to retrieve.

tables

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

WHERE conditions

Optional. These are conditions that must be met for the records to be selected.

Note

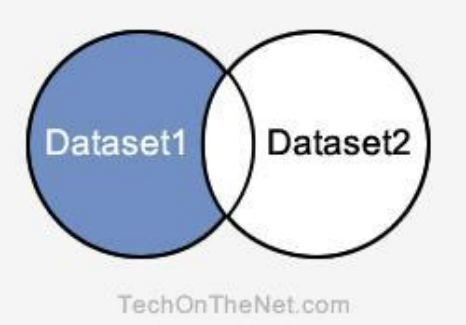
- There must be same number of expressions in both SELECT statements.
- The corresponding expressions must have the same data type in the SELECT statements. For example: *expression1* must be the same data type in both the first and second SELECT statement.

Except Operator:-

Description

The SQL Server (Transact-SQL) EXCEPT operator is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement. Each SELECT statement will define a dataset. The EXCEPT operator will retrieve all records from the first dataset and then remove from the results all records from the second dataset.

Except Query:-



Explanation: The EXCEPT query will return the records in the blue shaded area. These are the records that exist in Dataset1 and not in Dataset2.

Each SELECT statement within the EXCEPT query must have the same number of fields in the result sets with similar data types.

Syntax

The syntax for the EXCEPT operator in SQL Server (Transact-SQL) is:

```
 ELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions]
EXCEPT
SELECT expression1, expression2, ... expression_n
FROM tables
[WHERE conditions];
```

Parameters or Arguments

expressions

The columns or calculations that you wish to compare between the two SELECT statements. They do not have to be the same fields in each of the SELECT statements, but the corresponding columns must be similar data types.

tables

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

WHERE conditions

Optional. The conditions that must be met for the records to be selected.

Note

- There must be same number of expressions in both SELECT statements.
- The corresponding columns in each of the SE ECT statements must have similar data types.
- The EXCEPT operator returns all records from the first SELECT statement that are not in the second SELECT statement.
- The EXCEPT operator in QL erver is equivalent to the MINUS operator in Oracle.

Nested Queries:-

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

ubqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow –

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.

- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows –

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name ]
   FROM table1 [, table2 ]
   [WHERE])
```

Example

Consider the CUSTOMERS table having the following records –

```
+----  +----------+-----  +-----------+.....          +
 | ID | NAME      | AGE | ADDRESS    | SALARY     |
+----  +----------+-----  +-----------+.....          +
|   1 | Ramesh    |   35 | Ahmedabad |   2000.00  |
|   2 | Khilan    |   25 | Delhi     |   1500.00  |
|   3 | kaushik   |   23 | Kota      |   2000.00  |
|   4 | Chaitali  |   25 | Mumbai    |   6500.00  |
|   5 | Hardik    |   27 | Bhopal    |   8500.00  |
|   6 | Komal     |   22 | MP        |   4500.00  |
|   7 | Muffy     |   24 | Indore     | 10000.00 |
+----  +----------+-----  +-----------+.....          +
```

Now, let us check the following subquery with a SELECT statement.

```
SQL> SELECT *
   FROM CUSTOMERS
   WHERE ID IN (SELECT ID
         FROM CUSTOMERS
         WHERE SALARY > 4500) ;
```

This would produce the following result.

```
                -----  ---------  ----------
+----+----------+     +         +           +
| ID | NAME     |AGE|   ADDRESS | SALARY    |
+----+----------+-----+---------+---------- +
|   4| Chaitali |   25| Mumbai  |   6500.00 |
|   5| Hardik   |   27| Bhopal  |   8500.00 |
|   7| Muffy    |   24| Indore  | 10000.00  |
                -----  ---------  ----------
+----+----------+     +         +           +
```

Subqueries with the INSERT Statement

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows.

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
   SELECT [ *|column1 [, column2 ]
   FROM table1 [, table2 ]
   [ WHERE VALUE OPERATOR ]
```

Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

```
SQL> INSERT INTO CUSTOMERS_BKP
   SELECT * FROM CUSTOMERS
   WHERE ID IN (SELECT ID
   FROM CUSTOMERS) ;
```

Subqueries with the UPDATE Statement:-

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows.

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> UPDATE CUSTOMERS
   SET SALARY = SALARY * 0.25
   WHERE AGE IN (SELECT AGE FROM CU TOMERS_BKP
      WHERE AGE >= 27 );
```

This would impact two rows and finally CUSTOMERS table would have the following records.

```
                 -----            +----------
+----+----------+     +-----------            +
| ID | NAME     | AGE | ADDRESS   | SALARY    |
                 -----            +----------
+----+----------+     +-----------            +
|   1| Ramesh   |   35| Ahmedabad |    125.00 |
|   2| Khilan   |   25| Delhi     |   1500.00 |
|   3| kaushik  |   23| Kota      |   2000.00 |
|   4| Chaitali |   25| Mumbai    |   6500.00 |
|   5| Hardik   |   27| Bhopal    |   2125.00 |
|   6| Komal    |   22| MP        |   4500.00 |
|   7| Muffy    |   24| Indore     | 10000.00 |
                 -----            +----------
+----+----------+     +-----------            +
```

Subqueries with the DELETE Statement:-

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows.

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> DELETE FROM CUSTOMERS
   WHERE AGE IN (SELECT AGE FROM CUST MERS_BKP
      WHERE AGE >= 27 );
```

This would impact two rows and finally the CUSTOMERS table would have the following records.

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS |  SALARY  |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

Aggregate Operators:-

An aggregate function allows you to perform a calculation on a set of values to return a single scalar value. We often use aggregate functions with theGROUP BY and HAVING clauses of the SELECTstatement.

The following are the most commonly used SQL aggregate functions:

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

Notice that all aggregate functions above ignore `NULL` values except for the `COUNT` function.

SQL aggregate functions syntax

To call an aggregate function, you use the following syntax:

```
1 aggregate_function (DISTINCT | ALL expression)
```

Let's examine the syntax above in greater detail:

- First, specify an aggregate function that you want to use e.g., `MIN`, `MAX`, `AVG`, `SUM` or`COUNT`.
- Second, put `DISTINCT` or `ALL` modifier followed by an expression inside parentheses. If you explicitly use `DISTINCT` modifier, the aggregate function ignores duplicate values and only consider the unique values. If you use the `ALL` modifier, the aggregate function uses all values for calculation or evaluation. The `ALL` modifier is used by default if you do not specify any modifier explicitly.

SQL aggregate function examples

Let's take a look some examples of using SQL aggregate functions.

COUNT function example

To get the number of the products in the `products` table, you use the `COUNT` function as follows:

```
1 SELECT
2    COUNT(*)
3 FROM
4    products;
```

	COUNT(*)
▶	77

More information on the COUNT function.

AVG function example

To calculate the average units in stock of the products, you use the AVG function as follows:

```
1 SELECT
2    AVG(unitsinstock)
3 FROM
4    products;
```

	AVG(unitsinstock)
▶	40.5065

To calculate units in stock by product category, you use the AVG function with the GROUP BYclause as follows:

```
1 SELECT
2    categoryid, AVG(unitsinstock)
3 FROM
4    products
5 GROUP BY categoryid;
```

	categoryid	AVG(unitsinstock)
▶	1	46.5833
	2	42.2500
	3	29.6923
	4	39.3000
	5	44.0000
	6	27.5000
	7	20.0000
	8	58.4167

More information on AVG function.

SUM function example

To calculate the sum of units in stock by product category, you use the `SUM` function with the `GROUP BY` clause as the following query:

```
1 SELECT
2    categoryid, SUM(unitsinstock)
3 FROM
4    products
5 GROUP BY categoryid;
```

	categoryid	sum(unitsinstock)
▶	1	559
	2	507
	3	386
	4	393
	5	308
	6	165
	7	100
	8	701

Check it out the SUM function tutorial for more information on how to use the `SUM` function.

MIN function example

To get the minimum units in stock of products in the products table, you use the MIN function as follows:

```
1 SELECT
2    MIN(unitsinstock)
3 FROM
4    products;
```

	MIN(unitsinstock)
▶	0

More information on the MIN function.

MAX function example

To get the maximum units in stock of products in the products table, you use the MAX function as shown in the following query:

```
1 SELECT
2    MAX(unitsinstock)
3 FROM
4    products;
```

	Max(unitsinstock)
▶	125

Check it out the MAX function tutorial for more information.

In this tutorial, we have introduced you to the SQL aggregate functions including the most commonly used functions: AVG, COUNT, MIN, MAX, and SUM.

NULL VALUES:-

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax

The basic syntax of **NULL** while creating a table.

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.

A field with a NULL value is the one that has been left blank during the record creation.

Example

The NULL value can cause problems when selecting data. However, because when comparing an unknown value to any other value, the result is always unknown and not included in the results. You must use the **IS NULL** or **IS NOT NULL** operators to check for a NULL value.

Consider the following CUSTOMERS table having the records as shown below.

```
+----  +----------+-----  +----------+          +
 | ID | NAME      | AGE | ADDRESS   | SALARY    |
+---- +----------+----- +----------+----------  +
|   1 | Ramesh   |   32 | Ahmedabad |   2000.00 |
|   2 | Khilan   |   25 | Delhi     |   1500.00 |
|   3 | kaushik  |   23 | Kota      |   2000.00 |
|   4 | Chaitali |   25 | Mumbai    |   6500.00 |
|   5 | Hardik   |   27 | Bhopal    |   8500.00 |
|   6 | Komal    |   22 | MP        |           |
|   7 | Muffy    |   24 | Indore    |           |
+----  +----------+-----  +----------+          +
```

Now, following is the usage of the **IS NOT NULL**operator.

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY

   FROM CUSTOMERS

   WHERE  ALARY IS NOT NULL;
```

This would produce the following result –

```
                -----           +----------
+----+----------+     +----------           +
| ID | NAME     | AGE | ADDRESS   | SALARY    |
                -----           +----------
+----+----------+     +----------           +
|   1| Ramesh   |   32| Ahmedabad |   2000.00 |
|   2| Khilan   |   25| Delhi     |   1500.00 |
|   3| kaushik  |   23| Kota      |   2000.00 |
|   4| Chaitali |   25| Mumbai    |   6500.00 |
|   5| Hardik   |   27| Bhopal    |   8500.00 |
                -----           +----------
+----+----------+     +----------           +
```

Now, following is the usage of the **IS NULL** operator.

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY

   FROM CUSTOMERS

   WHERE SALARY IS NULL;
```

This would produce the following result –

```
                -----
+----+----------+     +-----------+---------- +
| ID | NAME     | AGE | ADDRESS   | SALARY    |
                -----
+----+----------+     +-----------+---------- +
|   6| Komal    |   22| MP        |           |
|   7| Muffy    |   24| Indore    |           |
+----+----------+-----+-----------+---------- +
```

Complex Integrity Constraints in SQL:-

Integrity Constraints are used to apply business rules for the database tables.

The constraints available in SQL are **Foreign Key**,**Not Null**, **Unique**, **Check**.

Constraints can be defined in two ways

1) The constraints can be specified immediately after the column definition. This is called column-level definition.

2) The constraints can be specified after all the columns are defined. This is called table-level definition.

1) SQL Primary key:

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

Syntax to define a Primary key at column level:

```
column name datatype [CONSTRAINT constraint_name] PRIMARY KEY
```

Syntax to define a Primary key at table level:

```
[CONSTRAINT constraint_name] PRIMARY KEY (column_name1,column_name2,..)
```

- **column_name1, column_name2** are the names of the columns which define the primary Key.
- The syntax within the bracket i.e. [CONSTRAINT constraint name] is optional.
  - **For Example:** To create an employee table with Primary Key constraint, the query would be like.
  - **Primary Key at column level:**
  - ```
    CREATE TABLE employee
    (  id  number(5)  PRIMARY KEY,
    name char(20),
    dept char(10),
    age number(2),
    salary number(10),
    location char(10)
    );
    ```
  - or
  - ```
    CREATE TABLE employee
    (  id  number(5)  CONSTRAINT  emp_id_pk  PRIMARY  KEY,
    name char(20),
    dept char(10),
    age number(2),
    salary number(10),
    location char(10)
    );
    ```

Primary Key at column level:

```
CREATE TABLE employee
(
id number(5),
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT emp id pk PRIMARY KEY (id)
);
```

Primary Key at table level:

```
CREATE TABLE employee
(
id number(5), NOT NULL,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
ALTER TABLE employee ADD CONSTRAINT PK_EMPLOYEE_ID PRIMARY
KEY (id)
);
```

Foreign key:-

A FOREIGN KEY is a key used to link two tables together.

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

The table containing the foreign key is called the child table, and the table containing the candidate 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	vendson	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 PR MARY KEY in the "Persons" table.

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

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

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

Active Database Concepts and Triggers:-

- Rules that specify actions that are automatically triggered by certain events have been considered important enhancements to database systems for quite some time. In fact, the concept of **triggers**—a technique for specifying certain types of active rules—has existed in early versions of the SQL specification for relational databases and triggers are now part of the SQL-99 and later standards.

Generalized Model for Active Databases and Oracle Triggers:-

The model that has been used to specify active database rules is referred to as the **Event-Condition-Action** (**ECA)** model. A rule in the ECA model has three components:

- 1. The **event(s)** that triggers the rule: These events are usually database update operations that are explicitly applied to the database. However, in the general model, they could also be temporal events[2] or other kinds of external events.

2. The **condition** that determines whether the rule action should be executed: Once the triggering event has occurred, an *optional* condition may be evaluated. If *no condition* is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only *if it evaluates to true* will the rule action be executed.

- 3. The **action** to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

SYNTAX:-

```
-- Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML
Trigger)
```

A syntax summary for specifying triggers in the Oracle system (main options only).

<trigger> ::= CRE TE TRIGGER <trigger name>

(AFTER I BEFORE) <triggering events> ON <table name>

[FOR EACH ROW]

[WHEN <condition>]

<trigger actions> ;

<triggering events> ::= <trigger event> {OR <trigger event> }

<trigger event> ::= INSERT I DELETE I UPDATE [OF <column name> { , <column name> }]

<trigger action> ::= <PL/SQL block>

Advantages

Followings are the advantages of using MySQL triggers:

1. **Integrity of data:** With the help of MySQL trigger we can check the integrity of data in the table. In other words, MySQL triggers are the alternative way to check the integrity of data.
2. **Useful for catching errors:** MySQL triggers can catch errors in business logic in the database layer.
3. **Alternative way to run scheduled tasks:** Actually by using MySQL triggers we do not have to wait to run the scheduled tasks because the triggers are invoked automatically 'before' or 'after' a modification is done to the data in the table.
4. **Auditing:** Actually MySQL triggers are very much useful for the purpose of auditing of the changes made in the table.
5. **Prevention of invalid transactions:** MySQL triggers are very useful in the prevention of invalid transactions.
6. **Logging of event:** MySQL triggers can log an event and can also store the information on the access of table.

Disadvantages

Followings are the disadvantages of using MySQL triggers:

1. **Cannot replace all validations:** Actually, MySQL triggers cannot replace all the validations and can only provide an extended validation.
2. **Invisible from client applications:** Basically MySQL triggers are invoked and executed invisible from the client applications hence it is very much difficult to figure out what happens in the database layer.
3. **Impose load on server:** Triggers can impose a high load on the database server.
4. **Not recommended for high velocity of data:** Triggers are not beneficial for use with high-velocity data i.e. the data when a number of events per second are high. It is because in case of high-velocity data the triggers get triggered all the time.