SQL

# **SQL :**

To extract the required information from a huge chunk of data by using queries.

## **PARTS OF A SQL QUERY:**

* **SELECT-**  fields from which the data needs to be retrieved.
* **FROM-** table from which the data is extracted
* **WHERE-** condition for extracting the data.
* **ORDER BY-** ascending/descending order in which the data needs to be arranged.

## **TO INSERT A COMMENT:**

The below screenshot represents the two way of inserting a comment in a database.

**Comments** are usually used to provide additional information.

A picture containing text, font, screenshot, line

Description automatically generated

# **TO CREATE A DATA BASE:**

Generally, we will create a database and then work on it by refreshing and clicking on the DBMS.

***Syntax: Create database database name;***

A screenshot of a computer

Description automatically generated

When you want to use that database, then you need not click the database every time instead, you can follow the below steps:

***Syntax****: use* ***database name****;*

## **TO CREATE A TABLE:**

***Syntax:***

Create table table\_name(

Column 1 datatype,

Column 2 datatype,

Column n datatype);

***Example:***

***A picture containing text, screenshot, font

Description automatically generated***

A picture containing text, screenshot, font, line

Description automatically generated

**What will happen if you run the same create table query?**

It will throw an error message.

A screenshot of a computer

Description automatically generated

**How to overcome this error?**

A white background with black text

Description automatically generated

Create table—once u created a table when you want to re-run the same table the use the above syntax.

The same applies to database as well. But the “if not exists” keyword will work for MySQL workbench. Incase of PostgreSQL, it will throw an error.

A screenshot of a computer

Description automatically generated

### **INSERTING VALUES IN A TABLE:**

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*, ...);

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:***

A picture containing text, font, screenshot, line

Description automatically generated

**A picture containing text, font, screenshot, line

Description automatically generated**

A screenshot of a computer code

Description automatically generated with low confidence

### **HOW TO CREATE A CUSTOM-MADE COLUMN NAME:**

#### **AS KEYWORD:**

When you want to output the result in a professional manner say, the data has a column ‘**Name’** and you’d like to output it as ‘**Emp Name’**, then follow the below steps:

***Syntax:***

SELECT NAME AS [EMP\_NAME] FROM CUSTOMS;

**[OR]**

SELECT NAME AS EMP\_NAME FROM CUSTOMS;

**[OR]**

SELECT NAME AS ‘EMP\_NAME’ FROM CUSTOMS;

A screenshot of a social media account

Description automatically generated with low confidence

A picture containing text, screenshot, font, line

Description automatically generated

***A screenshot of a computer

Description automatically generated with medium confidence***

### **LIMIT QUERY RESULTS BY USING “LIMIT” CLAUSE:**

When there are 100s of records in a table and you want to display only few records, then you use **“limit”** clause.

***Syntax:***

SELECT COLUMN 1,COLUMN 2 FROM TABLE\_NAME ORDER BY COLUMN 1 ASC/DESC LIMIT N;

***Example:***

## **TO UPDATE A TABLE:**

By using “update” keyword, you can update specific records or the entire table.

***Syntax:***

**UPDATE** table name.

**Set** column1= value1, column2=value2, column=value n

**WHERE** condition.

***Example:***

**UPDATE** Expo air

**Set** name=’harini’, age= 23

Where emp id = ‘GSC442’;

**Note:**

* If you do not set a condition for updating a table(do not have where condition), then it will update the entire table.

**Example:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## **TO DELETE A TABLE:**

### **TO DELETE THE RECORDS OF A TABLE:**

**Syntax:**

**DELETE FROM** table name **WHERE** condition;

**Example:**

**DELETE FROM** expo air **WHERE** emp name = ‘aravind’

### **TO DELETE ALL THE RECORDS OF A TABLE:**

**Syntax:**

DELETE FROM table\_name;

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

**Example**

DELETE FROM Customers;

### **TO DELETE THE ENTIRE TABLE:**

**Syntax:**

**DROP** **TABLE** table name;

**Example:**

**DROP TABLE** customers;

## **SQL OPERATORS:**

### **SELECT AND FROM CLAUSE:**

The **SELECT** statement is used to return values.

***Syntax:***

SELECT \*FROM TABLE\_NAME;

**SELECT** column1, column2, ...

**FROM** table\_name;

***Example:***

***A screenshot of a computer

Description automatically generated with medium confidence***

A screenshot of a computer

Description automatically generated with medium confidence

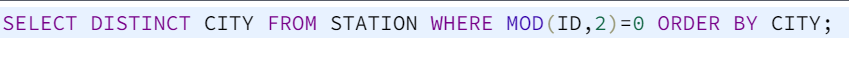
### **DISTICT CLAUSE:**

**“Distinct”** clause is used to remove the duplicates from the table.

Here **MOD** function is used to find the even number of IDs in the table.

**Example:**

**The below example represents the unique city names that has even ID numbers.**



A screenshot of a computer

Description automatically generated

### **TOP CLAUSE:**

* The SELECT TOP clause is used to specify the number of records to return.
* The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

**Example:** Select TOP N from Table\_name;

### **WHERE CLAUSE:**

* **‘Where’** clause is just a condition to refine data.
* **‘Where’** clause is used to refine only aggregate data.
* For instance, In a table that contains the below data, if they are asking you to find the Customer first name starting with ‘J’, then you can sort it in ascending order and can manually count the records in case of a small table when there is huge data you use ‘where’ clause to refine the data based on some conditions.
* **Note**: The WHERE clause is not only used in SELECT statements, but also in UPDATE, DELETE, etc.!

***Syntax:***

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

***Example:***

***A screenshot of a computer

Description automatically generated with medium confidence***

#### **TEXT FIELDS VS. NUMERIC FIELDS:**

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

**Example:**

SELECT \* FROM Customers  
WHERE CustomerID = 1;

SELECT \* FROM Customers  
WHERE Country = 'Mexico';

### **TO USE MORE THAN ONE CONDITION:**

#### **AND OPERATOR:**

This simply means to use more than one ‘**where’ clause.**

***Example:***

A screenshot of a computer

Description automatically generated with medium confidence

#### **OR OPERATOR:**

**Or** operator is used to output the result when either of the condition is true.

***Example:***

***A screenshot of a computer

Description automatically generated with medium confidence***

#### **NOT OPERATOR:**

This is a negation operator. It returns the opposite of the condition passed.

**Syntax:**

SELECT column FROM table\_name where NOT column<condition>;

**Example:**

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

### **ORDER BY CLAUSE:**

**Order By** clause is used to sort the data either in ascending or descending order.

The default sort is always **ascending.**

***Syntax:***

SELECT COLUMN 1,COLUMN 2…COLUMN N FROM TABLE\_NAME ORDER BY COLUMN 2 ASC;

SELECT COLUMN 1,COLUMN 2…COLUMN N FROM TABLE\_NAME ORDER BY COLUMN 2 DESC;

***Example:***

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

#### **ORDER BY SEVERAL COLUMNS EXAMPLE:**

* The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

**Example**

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

* The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:

**Example**

SELECT \* FROM Customers  
ORDER BY Country ASC, CustomerName DESC;

### **NULL OPERATOR:**

The “NULL” operator is used to fetch the records which are left blank in a table. When I mean “blank”, It refers to the records which do not have any values. It do not includes blank spaces or zeros.

**Note:** Null operators cannot be used with logical operators like ‘=’ or ‘!=’. It is used as “IS NULL” or “IS NOT NULL” key word.

**Syntax(IS NULL):**

SELECT column1,column2,column3 FROM table name WHERE column2 IS NULL;

*Note: This fetches the values of the columns(1,2 and 3) only when the column 2 does not have a value or is left blank.*

**Example:**

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NULL;

**Syntax(IS NOT NULL):**

SELECT column1,column2,column3 FROM table name WHERE column2 IS NOT NULL;

*Note: This fetches the values of the columns(1,2 and 3) only when the column 2 does not have a value or is not left blank.*

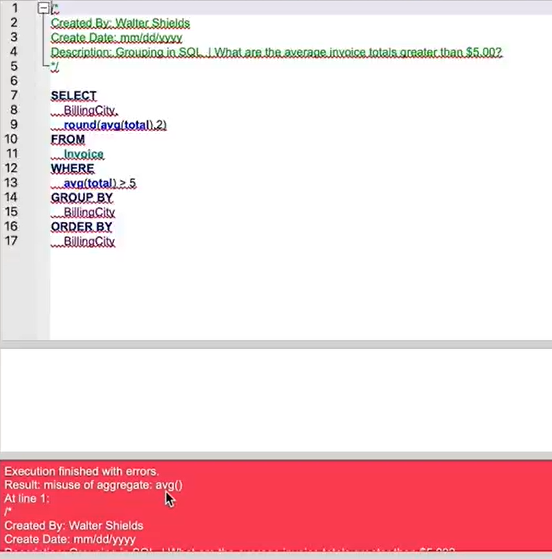
**Example:**

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NOT NULL;

### **HAVING CLAUSE:**

* Having clause is used as a secondary filter condition instead of where clause
* Having clause is used to filter aggregate function.
* Having clause should always come after **Group By clause.**

Say, Consider the below example where they use where clause to refine a aggregate condition. In this case, the system throws an error and **HAVING** clause comes into action.



Upon using **HAVING** clause, You can filter a non-aggregate function(used in GROUP BY) by a aggregate condition

A screenshot of a computer

Description automatically generated

### **USE OF BOTH WHERE AND HAVING CLAUSE:**

A screenshot of a computer

Description automatically generated

### **GROUPING BY MANY FIELDS:**

In the case where you need to group the query by using many fields, You can simply separate the column names by commas and use it in the query

A screenshot of a computer

Description automatically generated

### **BETWEEN OPERATOR:**

This operator returns the result between any specific range.

***Syntax:***

SELECT column1, column2, ...

FROM table\_name

WHERE Condition(uses between column 1 and column 2);

***Example:***

***A screenshot of a computer

Description automatically generated***

### **IN OPERATOR:**

**In** operator is like that of ‘**OR’** operator. The below example provides a list of employee names who are 22,25 and 40 age.

***Example:***

***A screenshot of a computer

Description automatically generated with medium confidence***

A screenshot of a computer

Description automatically generated with medium confidence

### **LIKE OPERATOR:**

* **Like** operator is used to refine data for uncertain conditions. Say, You want to filter the last name that starts with ’L’, then like operator is used.
* This operator can be used with a **wild card operator %**
* **%** operator simply means it ignores whatever that is going to come when the operator is used.
* **Example: L%** means it will consider all the records that starts with ‘**L’**

***Example:***

***A screenshot of a computer

Description automatically generated with medium confidence***

A screenshot of a computer

Description automatically generated with medium confidence

### **UNION OPERATOR:**

This operator is used to combine two different SQL queries.

**Example:**

**The below example illustrates to query city with minimum and maximum characters by eliminating the duplicates.**

A screenshot of a computer code

Description automatically generated with low confidence

A black rectangle with white text

Description automatically generated with low confidence

### **GROUP BY:**

This is used when you want to build a aggregate and non-aggregate field in the query. When you don’t use a group by in this scenario, You will get only one output instead of getting step by step output.

**QUERY WITHOUT GROUP BY CLAUSE:**

A screenshot of a computer

Description automatically generated

**QUERY WITH GROUP BY CLAUSE:**

A screenshot of a computer

Description automatically generated

### **DATES:**

* **Date** works the same way like other operators.
* You can use date function when the date contains both date and time
* Date function should be included as a keyword followed by parenthesis that has the field that contains date in the data.

***Example:***

***A screenshot of a computer

Description automatically generated with low confidenceA screenshot of a computer

Description automatically generated with low confidence***

### **BRACES AND ORDER:**

This is just like BODMAS or PEDMAS rule when there are more than one conditions used on a large amount of data. Without parenthesis, the conditions won’t be applied.

***Example:***

The below example is applied without parenthesis which gives us a wrong output as the first condition > 1.98 is applied only for Billing city like P% and not for the second condition.

A screenshot of a computer

Description automatically generated

Upon using the parenthesis, the first condition is applied to the billing city as well.

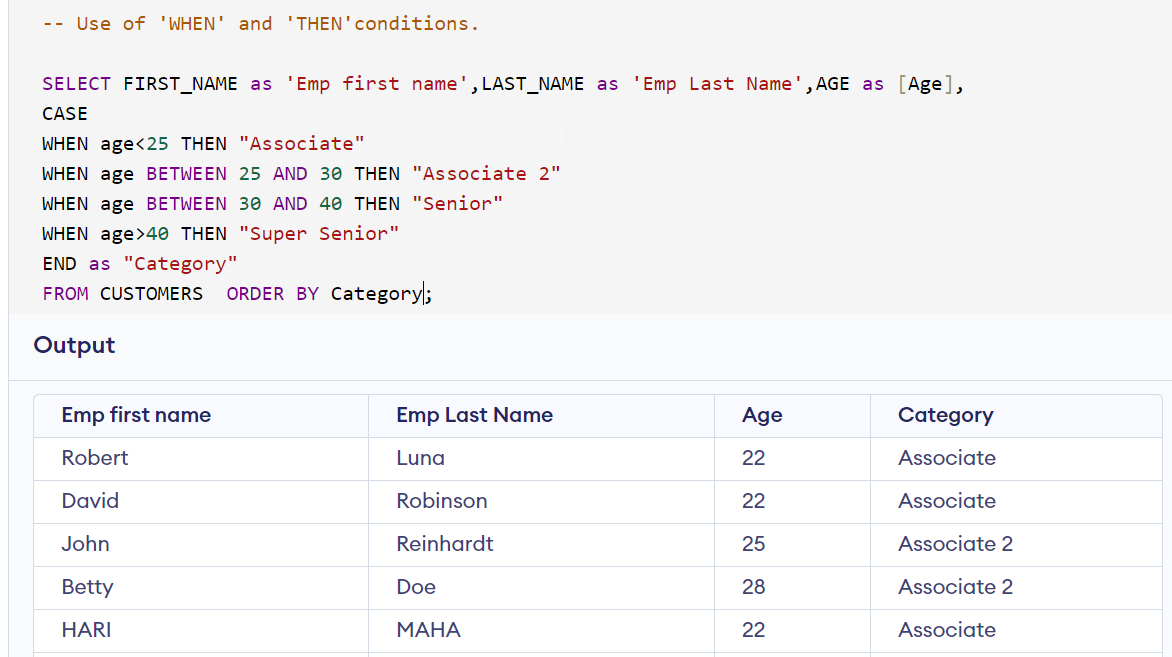
A screenshot of a computer

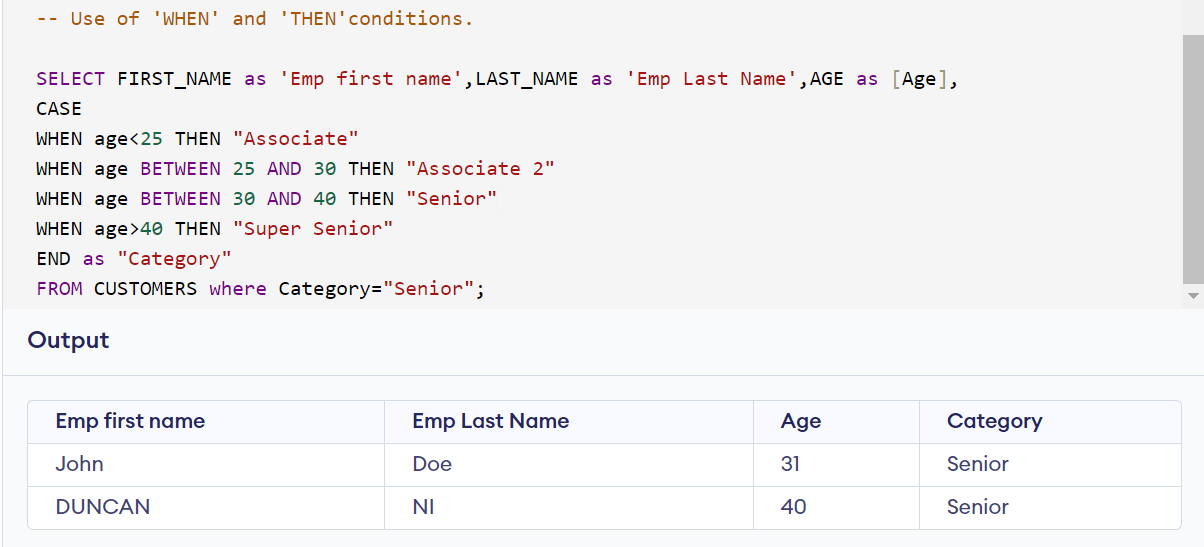
Description automatically generated

### **WHEN AND THEN CLAUSE:**

**When** and **Then** clause are like if and output values. If condition is true, then output some value.

***Example:***

******



# JOINS:

Joins are used to combine data from two tables.

## **INNER JOINS:**

***Syntax:***

SELECT column1,column2..column n FROM table\_name1

INNER JOIN table\_name2

ON table\_name1.foreign\_key=table\_name2.foreign\_key

***Example:***

***A screenshot of a computer

Description automatically generated with low confidence***

***Output:***

***A screenshot of a computer

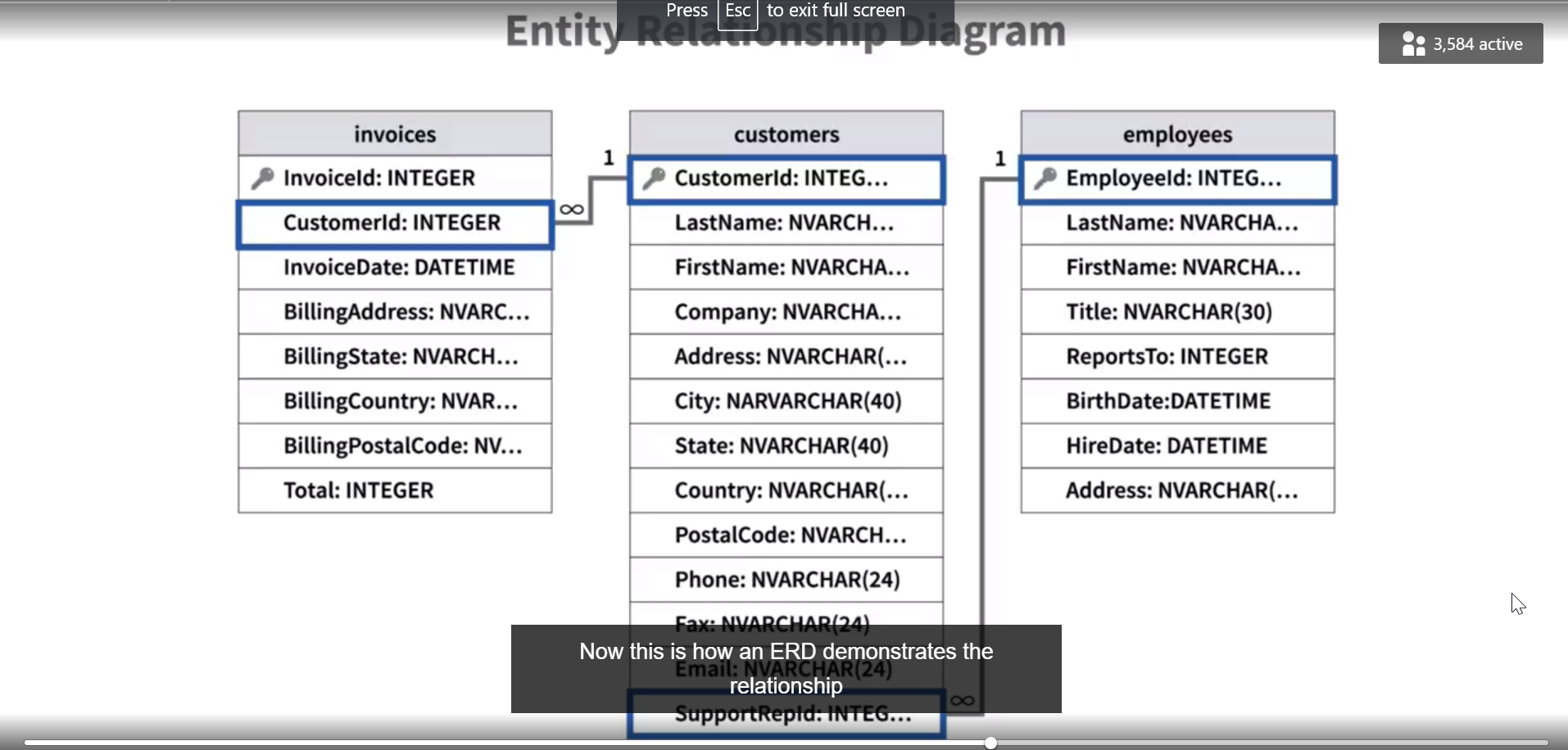
Description automatically generated***

***The above query will result in combining the fields of both the tables with the help of foreign key(i.e, customer\_id)***

## **PRIMARY KEY AND FOREIGN KEY:**

**Primary key:** A field that uniquely identifies the records in a table. Please find the screenshot below where the ***invoice\_id*** is the unique key of invoice table.

**Foreign key:** A field that is common between both the table and the primary key in any one of those tables. Here ***customer\_id*** is referred to us as the foreign key as it is found in both customer and invoice tables.



## NORMALIZATION:

Instead of using joins to combine the data from different tables, We can create it as a whole table with huge amount of data. The reason behind creating separate tables with relevant fields is to:

* Remove duplicates from the table
* Reduce the processing time for queries

## SIMPLYFING JOINS USING ALIASES:

Instead of writing long queries u can use **“as” clause** to given alternative name for the table. This is only used in joins to reduce the time for constructing a qur=ery.

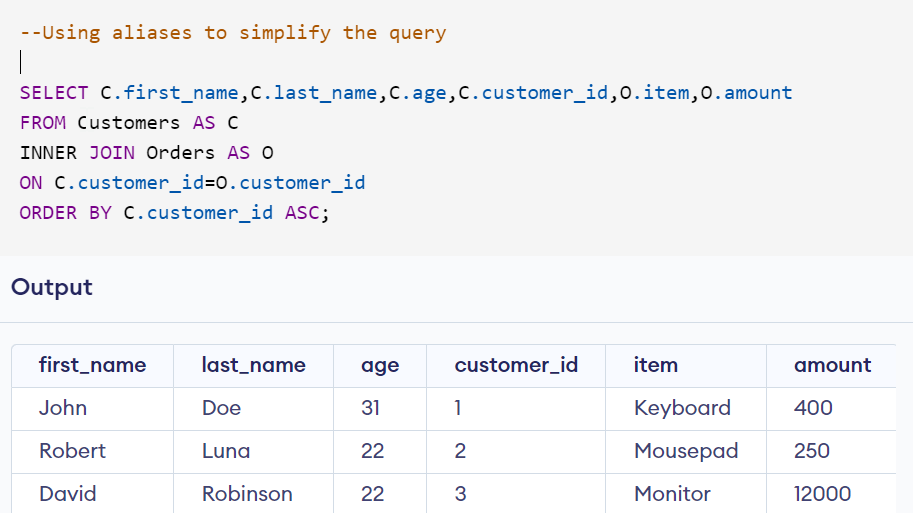
***Syntax:***

SELECT column1,column2..column n FROM table\_name1 AS T1

INNER JOIN table\_name2 AS T2

ON T1.foreign\_key=T2.foreign\_key

***Example:***

******

## DIFFERENCE BETWEEN INNER JOIN,LEFT OUTER AND RIGHT OUTER JOIN:

|  |  |  |
| --- | --- | --- |
| **INNER JOIN** | **LEFT OUTER JOIN** | **RIGHT OUTER JOIN** |
| The inner join is used to combine only the common fields between both the tables. All the unmatching fields will be ignored while using inner joins. | Left outer join will is used to combine two tables and all the data present in the left table will be combined while only the matching/relevant data of the right table will be taken into consideration. | Right outer join will is used to combine two tables and all the data present in the right table will be combined while only the matching/relevant data of the right table will be taken into consideration. |
| Inner join eliminates the data if they don’t match | Left join returns the data present in the left table no matter if it is matching or not but ignores the unmatched data in the right table. | Right join returns the data present in the right table no matter if it is matching or not but ignores the unmatched data in the right table |
| For example, In a scenario that has both invoice and customer table, And you are asked to find a list of customers who did not make any purchase for the past 5 days. Here **customer id is the foreign key.** There are 20 customer id’s in customer table while the invoice table has only 12 customer ids. The result produced will have only 12 customer id data as the SQL inner join ignores the remaining 8 ids that does not match. | For example, In a scenario that has both invoice and customer table, And you are asked to find a list of customers who did not make any purchase for the past 5 days. Here **customer id is the foreign key.** Here the customer table is left and invoice table is right.There are 20 customer id’s in customer table while the invoice table has only 12 customer ids. The result produced will have all the 20 customer id data but will have null values in the fields of those right 8 customers as the SQL left outer join fills null in the remaining 8 ids that does not match. | For example, In a scenario that has both invoice and customer table, And you are asked to find a list of customers who did not make any purchase for the past 5 days. Here **customer id is the foreign key.** Here the customer table is left and invoice table is right.There are 20 customer id’s in customer table while the invoice table has only 12 customer ids. The result produced will have all the 20 customer id data but will have null values in the fields of those left 8 customers as the SQL right outer join fills null in the remaining 8 ids that does not match. |
| No match fields are not displayed in the output. | Null values are filled in the field that do not find a match. | Null values are filled in the field that do not find a match. |
| **Syntax:**  SELECT column1,column2..column n FROM table\_name1  INNER JOIN table\_name2  ON table\_name1.foreign\_key=table\_name2.foreign\_key | **Syntax:**  SELECT column1,column2..column n FROM table\_name1  LEFT OUTER JOIN table\_name2  ON table\_name1.foreign\_key=table\_name2.foreign\_key | **Syntax:**  SELECT column1,column2..column n FROM table\_name1  RIGHT OUTER JOIN table\_name2  ON table\_name1.foreign\_key=table\_name2.foreign\_key |
| **Example:** | **Example:** | **Example:** |

A diagram of a table

Description automatically generated

#### INNER JOIN EXAMPLE:

A screenshot of a computer

Description automatically generated with medium confidence

**The below example shows the result of sum of population from two tables where the continent is “Asia”**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

#### **LEFT OUTER JOIN EXAMPLE:**

A screenshot of a computer

Description automatically generated with medium confidence

## TO JOIN THREE OR MORE TABLE:

***Example:***

***A screenshot of a computer code

Description automatically generated with low confidence***

***Output:***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Age** | **Customer\_Id** | **Item** | **Amount** | **Shipping\_Id** | status | | --- | --- | --- | --- | --- | --- | | John | Doe | 31 | 1 | Keyboard | 400 | 5 | Delivered | | Robert | Luna | 22 | 2 | Mousepad | 250 | 1 | Pending | | David | Robinson | 22 | 3 | Monitor | 12000 | 3 | Delivered | | John | Reinhardt | 25 | 4 | Keyboard | 400 | 2 | Pending | | John | Reinhardt | 25 | 4 | Mouse | 300 | 2 | Pending | |

## FULL OUTER JOIN:

Full outer join is used to extract records only if there is a match found in both right and left tables.

**Syntax:**

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition;

**Example:**

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

## SELF JOIN:

Self Join is used to join the items of the same table. Say you wanted to find the list of customers who are of same age.

Now in this case unlike other situations, you do not know the age that you need to match but the only condition is to divide the customers into two group and both should be of same age.

**Example:**

SELECT A.Customer Name, B.Customer name, A. Age

FROM Customer A , Customer B

WHERE A.Customer ID <> B.Customer ID

AND A.Age= B. Age

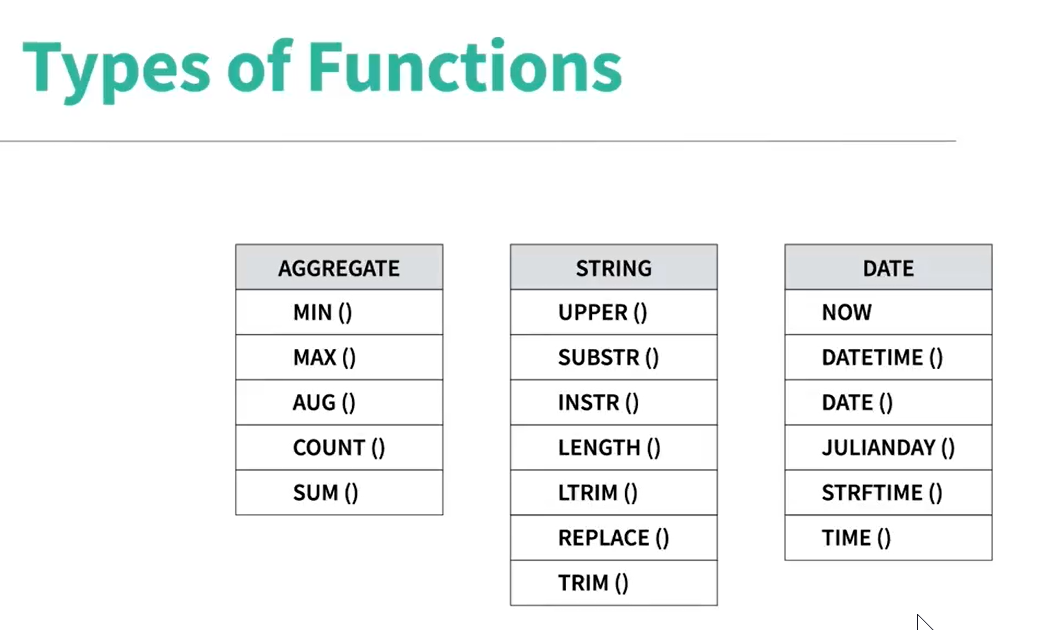
ORDER BY A. AGE

# FUNCTIONS:

## SQL AGGREGATE FUNCTIONS:

* An aggregate function is a function that performs a calculation on a set of values, and returns a single value.
* Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.
* The most commonly used SQL aggregate functions are:
* MIN() - returns the smallest value within the selected column
* MAX() - returns the largest value within the selected column
* COUNT() - returns the number of rows in a set
* SUM() - returns the total sum of a numerical column
* AVG() - returns the average value of a numerical column

Aggregate functions ignore null values (except for COUNT()).



### MOD FUNCTION

### COUNT():

**Count()** is used to count the number of of values present in a column of a table.

**Example:**

**The below example returns the difference between the total cities and the distinct cities in the Station table.**

A screenshot of a computer

Description automatically generated

### LENGTH():

Length() is used to find the length of the string.

**Example:**

**The below example illustrates the length of the city with minimum and maximum characters.**

**A screenshot of a computer code

Description automatically generated with low confidence**

**A black rectangle with white text

Description automatically generated with low confidence**

### LEFT AND RIGHT():

Left() and Right() is used to return the left most and right most characters of the string.

**Syntax:**

LEFT(STR,N)

STR-STRING NAME

N- NO.OF CHARACTERS

RIGHT(STR,N)

STR-STRING NAME

N- NO.OF CHARACTERS

**Example:**

**The below examples demonstrates the usage of left() and right() to sort the city names ending with vowels.**

**A close-up of a sign

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated**

A screen shot of a computer

Description automatically generated with low confidence

**A screenshot of a computer

Description automatically generated**

### SUBSTR():

The **substr()** is used to derive a character or a set of characters from a string.

**Syntax:**

**SUBSTR(STR,POS,N)**

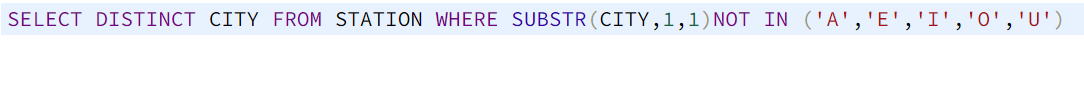
STR-STRING NAME

POS- STARTING POSISTION.

N- NO.OF CHARACTERS

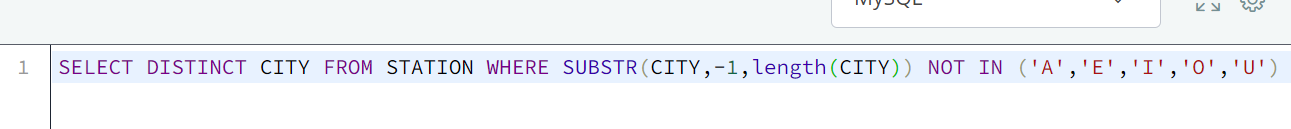
**Example:**

**The below example demonstrates the use of SUBSTR() to find the city names that do not start and end with vowels.**

****

**A screenshot of a computer

Description automatically generated**



A screenshot of a computer

Description automatically generated

### AVG() AND ROUND():

**Avg()** is used to find the average of number.

**Round()-** is used to find the rounded value.

**Syntax:**

**AVG(N)**

**N-** Number

**Example:**

**The below example demonstrates the rounded value of average population in a city.**

A picture containing text, screenshot, font, line

Description automatically generated

**A picture containing text, font, screenshot, graphics

Description automatically generated**

### SUM():

**Sum()** is used to add the values.

**Syntax:**

**SUM(N)**

**N-**Number

**Example:**

**The below example demonstrates the sum of population in Japan.**

**A picture containing text, screenshot, font, line

Description automatically generated**

### CEILING():

The **CEILING()** function returns the smallest integer value that is larger than or equal to a number.

Say 23.42 will be ceiled as 24.

**Syntax:**

**CEIL(N)**

**N-** Number

### REPLACE():

The REPLACE() function replaces all occurrences of a substring within a string, with a new substring.

**Syntax:**

**REPLACE(STR,Old string, New string)**

**STR-** String

**Old String-** What to be replaced.

**New String-** New string by which it will be replaced.

**Example:**

**In the below example, we find the difference between the actual average of salaries and the miscalculated salary where we missed ‘0’ while calculation.**



A screenshot of a computer

Description automatically generated with low confidence

### STUFF():

The **STUFF()** function deletes a part of a string and then inserts another part into the string, starting at a specified position.

**Syntax:**

**STUFF(***string*, *start*, *length*, *new string*)

|  |  |
| --- | --- |
| ***string*** | Required. The string to be modified |
| ***start*** | Required. The position in *string* to start to delete some characters |
| ***length*** | Required. The number of characters to delete from *string* |
| ***new\_string*** | Required. The new string to insert into *string* at the *start* position |

**Example:**

A screenshot of a computer program

Description automatically generated with low confidence

### TRUNCATE():

**Truncate()** is similar to a round off function which cuts down the value upto certain decimal places.

**Syntax:**

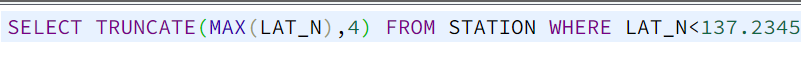
**TRUNCATE(NUM,N)**

**NUM-** Number to be truncated.

**N-** Decimal places upto which it should be truncated

**Example:**

**In the below example, The max of lat\_n is truncated to four decimal places.**

****

**A screenshot of a computer

Description automatically generated with medium confidence**

### ROUND():

**Round()** is used to round off the integer.

**Syntax:**

**ROUND(NUM,N)**

**NUM-**Number that needs to be rounded.

**N-**numbers after the decimal places

**Syntax:**

**In the below example, the average sale is rounded off two decimal places.**

**A picture containing text, software, web page, screenshot

Description automatically generated**

**In the below example, we have rounded off the sum of lat\_n and long\_w upto 2 decimal places**

A close-up of a computer screen

Description automatically generated with low confidence

A picture containing text, font, screenshot, graphics

Description automatically generated

### CONCAT():

* Concatenate two strings.
* **||** symbol can also be used to combine two columns.
* Say, First name is Harini and Last name is Mahalingam then it is concatenated as **HariniMahalingam (i.e. First name||Last name)**

### REPEAT():

This function is used to repeat a word or a letter or a string.

**Example:**

**A screenshot of a computer

Description automatically generated**

**Output:**

**A screenshot of a test

Description automatically generated**

### DATE():

#### DATE ADD :

DATE\_ADD() function adds a time/date interval to a date and then returns the date.

**Syntax:** DATE\_ADD(*date*, INTERVAL *value addunit*)

A screenshot of a computer

Description automatically generated

#### DATE SUB:

DATE\_SUB() function subtracts a time/date interval from a date and then returns the date

**Syntax:** DATE\_SUB(*date*, INTERVAL *value interval*)

A screenshot of a computer

Description automatically generated

#### DATEDIFF:

DATEDIFF() function returns the number of days between two date values.

**Syntax:**DATEDIFF(*date1*, *date2*)

**A screenshot of a computer

Description automatically generated**

#### STRFTIME:

**Strftime()** is used to convert he date time format into another format.

**Syntax:**

**Strftime(Format,Column)**

**Format-** This refers to the format in which you want to convert the column. Say YYYY-MM-DD can be written as %Y-%M-%D.

**Column-**Column that needs to be formatted.

**Example:**

**The below example demonstrates how the age has been calculated by using the strftime().**

**Now()- is used to calculate the current date time.**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Notes:**

1. **UNION ALL –** take all the values and not distinct values.
2. **UNION –** take only distinct values.

**QN**: A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **A screenshot of a computer

   Description automatically generated**
2. **How to check if the value is preset in a column or data?**

You can either use CHARINDEX() or IN keyword or LIKE keyword. Please refer to the below example for your reference:

A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated
2. A screenshot of a computer

   Description automatically generated
3. A screenshot of a computer

   Description automatically generated
4. A screenshot of a computer

   Description automatically generated
5. A screenshot of a computer

   Description automatically generated
6. A screenshot of a computer

   Description automatically generated
7. A screenshot of a computer

   Description automatically generated
8. A screenshot of a computer

   Description automatically generated
9. A screenshot of a computer

   Description automatically generated
10. A screenshot of a computer

    Description automatically generated
11. A screenshot of a computer

    Description automatically generated
12. A screenshot of a computer

    Description automatically generated
13. A screenshot of a computer

    Description automatically generated
14. A screenshot of a computer

    Description automatically generated
15. A screenshot of a computer

    Description automatically generated
16. A screenshot of a computer

    Description automatically generated
17. A screenshot of a computer

    Description automatically generated
18. A screenshot of a computer

    Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated
2. A screenshot of a computer

   Description automatically generated
3. A screenshot of a computer

   Description automatically generated

A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated

The above code was executed in **MS SQL Server**. Please find the code below which was executed in **MY SQL.**

A screenshot of a computer

Description automatically generated

1. A screenshot of a computer

   Description automatically generated

Output:

A screenshot of a computer

Description automatically generated

with grading as(

select name,

case

when marks>=90 then '10'

when marks between 80 and 89 then '9'

when marks between 70 and 79 then '8'

when marks between 60 and 69 then '7'

when marks between 50 and 59 then '6'

when marks between 40 and 49 then '5'

when marks between 30 and 39 then '4'

when marks between 20 and 29 then '3'

when marks between 10 and 19 then '2'

when marks between 0 and 9 then '1'

end as "grade",marks

from students

)

(select name,grade,marks from grading where grade=10

order by students.name)

UNION

(select name,grade,marks from grading where grade=9

order by name)

UNION ALL

(select name,grade,marks from grading where grade=8

order by name)

UNION ALL

(select 'NULL' as name,grade,marks from grading where grade=7

order by marks asc)

UNION ALL

(select 'NULL' as name,grade,marks from grading where grade=6

order by marks asc)

UNION ALL

(select 'NULL' as name,grade,marks from grading where grade=5

order by marks asc)

UNION ALL

(select 'NULL' as name,grade,marks from grading where grade<5

order by marks asc)

with grading as(

select name,

case

when marks>=90 then '10'

when marks between 80 and 89 then '9'

when marks between 70 and 79 then '8'

when marks between 60 and 69 then '7'

when marks between 50 and 59 then '6'

when marks between 40 and 49 then '5'

when marks between 30 and 39 then '4'

when marks between 20 and 29 then '3'

when marks between 10 and 19 then '2'

when marks between 0 and 9 then '1'

end as "grade",marks

from students

)

WITH combine as(

select \* from (

Select name,grade,marks from grading where grade>=8

UNION ALL

Select "NULL" as name,grade,marks from grading where grade<8))

select name,grade,marks from combine ORDER BY grade desc

with grading as(

select name,

case

when marks>=90 then '10'

when marks between 80 and 89 then '9'

when marks between 70 and 79 then '8'

when marks between 60 and 69 then '7'

when marks between 50 and 59 then '6'

when marks between 40 and 49 then '5'

when marks between 30 and 39 then '4'

when marks between 20 and 29 then '3'

when marks between 10 and 19 then '2'

when marks between 0 and 9 then '1'

end as "grade",marks

from student ` 15 s

)

select \* from (

Select name,grade,marks from grading where grade>=8)

UNION ALL

Select "NULL" as name,grade,marks from grading where grade<8) as T

ORDER BY grade des

* **Aamina Ashley Christeen Eve**
* **Julia Belvet Jane Jennifer**
* **Priya Britney Jenny Ketty**
* **NULL Maria Kristeen Samantha**
* **NULL Meera NULL NULL**
* **NULL Naomi NULL NULL**
* **NULL Priyanka NULL NULL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **10308** | **2** | **7** | **9/18/1996** | **3** |
| **10309** | **37** | **3** | **9/19/1996** | **1** |
| **10310** | **77** | **8** | **9/20/1996** | **2** |