SQL NOTES

Database : collection of data like Collection of information of employees ,students

Database types: 1.relational database /SQL database

2.non relational database /Non SQL database

Relational database:

* Stores information in tables
* Tables is a set rows and columns
* Have fixed schema
* Examles
  + - Oracle
    - Postgres
    - Sqllite
    - MySQL
    - MYSQL server

No-SQL Database:

Mainly used in huge amount of data

* Key value
* Document
* Graphs
* Examples:
  + - Mongo DB
    - Cassendra

SQL is common for all databases

Structured query language

Creating table

Syntax:

Create table employee(id Int, name varchar ,address varchar)

Adding values to it

Insert into employee(here we have to mention which values we are adding , if we are adding all values we don’t want to mention name, address ) here w are adding only 2 values

For adding required values

Insert into employee(name ,address)

Values(“name”,”hyd”)

For adding all values

Insert into employee

Values(1,“name”,”hyd”)

We can add multiple rows

Insert into employee

Values(1,“saketh”,”hyd”),(2,”venky”,”hyd”)

Select emp\_name as name from employee; It is only change emp\_name as name while selecting but it doesn’t change in database

Delete from table : it deletes the data from the table

Drop table employee: it deletes the table

Decimal(5(before decimal vaues),2(after decimal values))

Ex:32321.10

Primary keys and foreign keys ;

|  |  |
| --- | --- |
| Primary key | Foreign key |
| No duplicates allowed |  |
| Null values not allowed |  |
| Uniquely identified | It take reference of another table |

Ex:

Here we have emp1 table

|  |  |  |  |
| --- | --- | --- | --- |
| Emp\_Id | name | age | location |
| 1 | saketh | 22 | hyd |
| 2 | patel | 23 | chennai |

We have second table emp2 table

|  |  |  |
| --- | --- | --- |
| Emp\_id | salary | hike |
| 1 | 10,000 | 10 |
| 2 | 20,000 | 20 |

Here by using foreign key we can refer employee 1 table

To give hike to employees we use emp\_ids for that because some of employees have same name

So

Create table emp2 (foreign key(emp\_id) reference emp1(emp\_id ), salary int , hike int)

Insert into

Values( 1,10000,10)

If we give emp id other than in emp1 table it will throw the error that is use of foreign key

Unique:

Here no duplicate values are allowed

create table citizen(citizen\_id int primary key,citizen\_name

varchar(100),aadhar\_no varchar(12) unique ,country varchar(10) default

'india');

insert into citizen(citizen\_id,citizen\_name,aadhar\_no)

values(4,'venky','8899999999');

we don’t need give country name again because it default take country value as “india”

in voter list we doestnt give id it will automatically increase the values

by using auto\_increment in MYSQL or identity(1,1 ) in sql server

|  |  |
| --- | --- |
| Primary key | Unique key |
| It doesn’t allow null values | It will allow null values |

create table voters(id int primary key auto\_increment not null,

voter\_id int unique not null,voter\_name varchar(100)not null ,

age int check(age>=18) ,mobile\_num varchar(10) unique,citizen\_id int,

foreign key(citizen\_id) references citizen(citizen\_id)) ;

check>=18 checks the age if the user provides the age less than 18 it throws the error

SQL CONSTRAINTS:

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

Date :

To take the date from user we have to give the datatype as date , datetime

To give the todays today we use current\_date or current\_timestamp function

Ex:

create table sales(sales\_id int primary key auto\_increment , product\_name

Varchar(100) ,sales\_date datetime, quantity int ,price int);

drop table sales;

insert into sales(product\_name,sales\_date ,quantity,price )

values("washing machine",current\_timestamp(),1,234.33);

Distinct :

Distinct means it will display only unique values

Select product\_name from sales;

o/p:

Product\_name

|  |
| --- |
| washing machine |
| laptop |
| mobile |
| mobile |

Here it will select all rows in table

But by using distinct only unique values are select like mobile is selected only one time

select distinct product\_name from sales;

o/p:

Product\_name

|  |
| --- |
| washing machine |
| laptop |
|  |

Mobile

TO ADD EXTRA COLUMNS TO THE TABLE

EX:

alter table sales add discount decimal(5,2);

To Delete extra added columns

Ex:

alter table sales drop column discount;

To modify the datatype of extra added column

ALTER TABLE

To add or delete or modify the given table we use alter table command

To add or delete constraints of columns also we use alter table

Add

alter table sales add qty\_available decimal(5,2);

delete

alter table sales drop column qty\_available;

modifying column datatype

alter table sales modify column qty\_available int;

we can also add constraints

alter table sales add qty\_available int unique not null;

Order by

To oder the data by asending or descending order we use order by clause

Ex:

select \* from sales order by price desc;

like we filter in ajio products of cost from low to high or high to low

filtering : by using where clause

select \* from sales where price>=4000;

select \* from sales where price between 2000 and 4000;

Like operator

Like means we will select the values by using any keyword

This is a case insensitive

Ex:

select \* from sales where product\_name like 'lap%';

not like operator

select \* from sales where product\_name not like '%lap%';

IN operator

We want to select only required products so then in operator is used

Ex:

select \* from sales where product\_name in('phone','laptop');

to select products other then phone and laptops

Not IN

select \* from sales where product\_name not in('phone','laptop');

Null values

Is null

select \* from sales where discount IS null;

not null

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

min

it will give the min value from the selected column

ex: select min(price) from sales;

GROUP BY

Group by means it will group same items together

This is also called as aggregations

Like group by product name means it will group the same product name items together

Ex: we have sales table the CEO wants how much quantity of products sold

Like washing machine 2 sold

And mobile 3 sold

Like that product and how much quantity they sold

So for that here we have group by product name then count the values

Id product\_name sales date quantity price

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | washing machine | 2024-04-25 10:27:16 | 1 | 234 |  |  |
| 2 | Laptop | 2024-04-25 10:28:31 | 1 | 23333 |  |  |
| 3 | mobile | 2024-04-25 10:29:16 | 1 | 3333 |  |  |
| 4 | mobile | 2024-04-25 10:44:00 | 1 | 3333 |  |  |
| 5 | mobile | 2024-04-25 10:53:05 | 1 | 3333 |  |  |
| 6 | washing machine | 2024-04-25 10:27:16 | 1 | 234 |  |  |

Ex:

select product\_name ,count(product\_name) from sales

group by product\_name;

2. ceo wants the total sale value

Product count sum

Sales value of mobile 3 9999

Sales value of washing machine similarily for all

Code;

select product\_name ,count(product\_name) ,sum(price) from sales

group by product\_name;

3.he wants average of sales value

select product\_name ,count(product\_name) ,sum(price),avg(price) from sales

group by product\_name;

here we will get values but columns names are empty so to keep column names we use aliasing

ALIASING

Aliasing means setting the column header names of table as our requirement

Easy understanding purpose

Nickname to values

Ex:

select product\_name ,

count(product\_name) as "units sold",

sum(price) as "total sales value",

avg(price) as 'average price' from sales

group by product\_name;

when ever we want to filter output after group by we use having clause

HAVING

It is used for filtering after group by

Ex:

select product\_name ,

count(product\_name) as "units sold",

sum(price) as "total sales value",

avg(price) as 'average price' from sales

group by product\_name

having sum(price)>500;

**JOINS**

The joins are used to join the rows from two or more tables by using relative keys, like foreign key

Inner join :

It is used to join the rows of two tables if they are in both table 1 and table 2

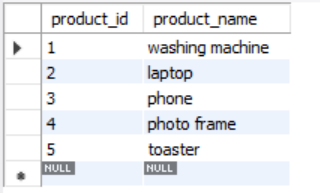
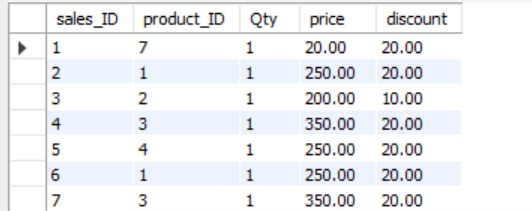
It will add common values in left table and right table

Ex:

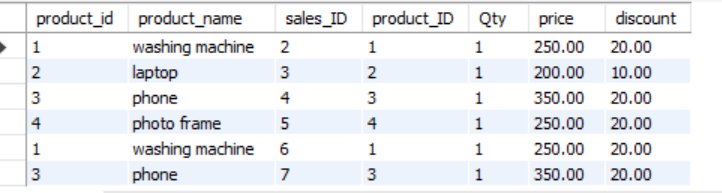
select \*from Products as P inner join sales1 as S

on P.product\_id=S.product\_ID

Left table right table

Inner join



Left join :

It will used to join the two tables it joins all values from left table but not from right table

It compares all left values with right table if matches with left table it will print all the values if not it will print only matched values

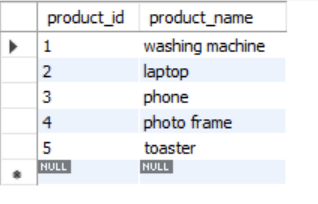
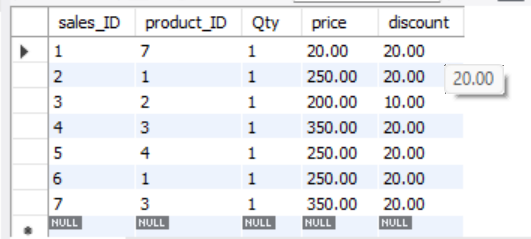
Ex:

Code:

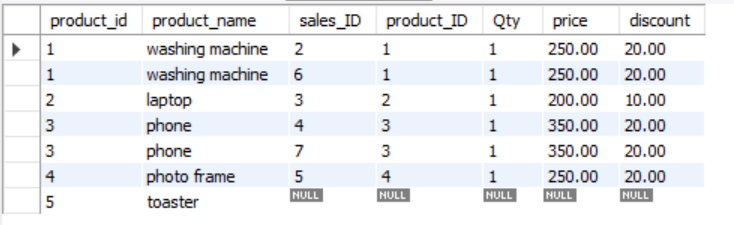
select \*from Products as P left join sales1 as S

on P.product\_id=S.product\_ID

left table Right table

Left join

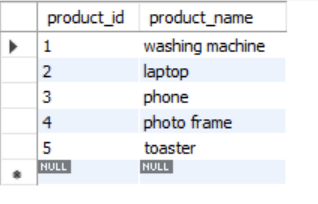
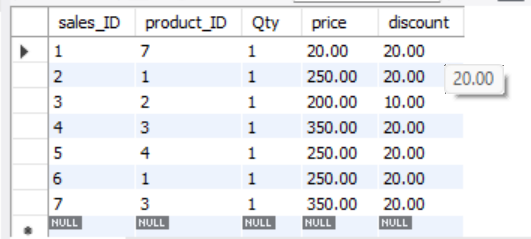


Right join:

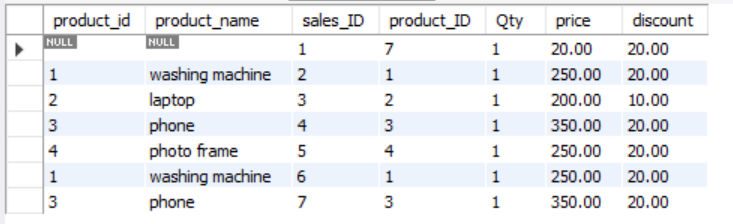
It join the two tables with all values from the right table to another table

It compares the the right table with left and it will print all values in right table and common values In left table

Left table right table

Right join



Full join/full outer join

It joins the all values in table left and table right

Self join

In one table only it will join

We have employee table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Employee-id | Emp-name | Dept | Manager-id |
| 1 | 4050 | saketh | IT | 4572 |
| 2 | 4572 | Arvind pandey | IT | 5920 |
|  |  |  |  |  |

Here it a employee table it shows employees of a company in that we have all employees like managers , freshers ,senior managers ,ceo

Here Saketh is a employee and Arvind pandey is also a employee in company but sakeeth has manager (2) it means id 2 we have Arvind pandey ,it means Arvind pandey is manger for Saketh

So to do this we use self join by matching manager id with emp- id

Ex:

Select \* from employee E Inner Join employee M on E.emp-id=M.manager-id

Here we are taking employee table as E one time and M one time



Declaring values in sql

Ex:

Select \* from sales ;

Declare @max\_price decimal(5,2) =0

Print ‘Initial value of sale price is ‘ +CAST (@max\_price as varchar)

Select @max\_price =MAX(price) from sales;

Print ‘maximum sale price is’+CAST(@max\_price as varchar)

CAST : it Is used for converting data types

Ex:

CAST(@max\_price as varchar)

If else in sql

IF @max\_price>400

BEGIN

Print ‘wow! Max sales for this month is ‘ CAST(@max\_price as varchar)

END

ELSE

BEGIN

Print ‘oops sales is not more than 400 ‘ +CAST(@max\_price as varchar)

END

Mysql if statement

select price ,if (max(price)>300,'more','less') from sales1

case

ex:

select student\_name ,

case

when score<=100 and score>=90 then ' Grade A'

when score<=89 and score>=70 then 'Grade B'

when score<=69 and score>=60 then 'Grade C'

when score<=59 and score>=50 then 'Grade D'

ELSE 'Grade E'

END as 'Grade'

from student\_scores;

here we using case for printing the grades of a student

@@identity

* it is used for storing the primary key value of table 1 to use in table 2
* normally we have 2 tables 1 is students basic details like student\_id ,name , mobile\_num,address
* table 2 have basic\_qualification details like 10th marks, 12th marks ,student\_ID is the foreign because its same in table 1 and table 2

we will insert values into table 1 like name and mobile ,address we not giving student\_ID because its auto increment

so while we enter values into table 2 we have to enter table 1 student\_id but we don’t know what is it

so for this by using identity we are storing the student\_Id while we enter to table 1

set @student\_ID =@@Identity;

it will store student\_ID after insering values to table 1 then we are using it while inserting values to table 2

insert into student\_qualifications(X\_marks,XII\_marks,student\_ID)

values(90,90,@student\_ID);

functions

COALESCE function

It returns the values inly and neglet null values

Ex: select coalesce(null,null,'sourabh',null);

o/p: sourabh

ABS function

It convert the –ve values to positive

like

Ex:

create table student\_basic\_ details (student\_ID int primary key auto\_increment,

student\_name varchar(100),mob varchar(10),address varchar(100));

create table student\_qualifications(qual\_id int primary key auto\_increment,

X\_marks INT,

XII\_marks int,

student\_ID int,

foreign key(student\_ID) references student\_basic\_details(student\_ID));

select \* from student\_basic\_details;

select \* from student\_qualifications;

Insert into student\_basic\_details(student\_name,mob,address)

values ('sourab','5587767','xyz');

set @student\_ID =@@Identity;

insert into student\_qualifications(X\_marks,XII\_marks,student\_ID)

values(90,90,@student\_ID);

select \*, ceil(price),floor(price),round(price,1) from sales1;

* ceil means it will convert the float values to int ex:235.77 to 236 any value to next number
* Floor converts float values to int ex: 235.77 to 235 any value to sme number
* Round converts round(235.77,1) to 235.8 235.22 to 235.2

select \*, adddate(insert\_date,interval 5 month) from customers;

* Add date is used to add months , days , years , to given date

Ex: select \*, adddate(2024-04-28, interval 5 month) ;

o/p: 2024-09-28

* Like this we can add days months years to given date

select datediff('2018/09/29','2022/09/29');

* It will print the date difference

select date\_format('2024-04-27','%m-%d-%y');

* We can change date format according to our requirement
* o/p: 04-27-2024

select coalesce(null,null,'sourabh',null);

* it will neglect null values
* o/p: sourabh

select abs(-9);

* it converts the –ve values to +values

we can assign column like this also

ex: select \* ,length(customer\_name) 'name length'

LEFT

* left (name ,how many values)
* it will print left side values from string
* ex: select left (‘saketh’,3)
* o/p: sak

RIGHT

* right (name ,how many values)
* it will print values from right side in string
* ex: select left (‘saketh’,3)
* o/p: eth

LENGHT

* it will print the length of the string

ex: select length(‘saketh);

UPPER

* it will convert given string to uppercase

ex: upper(‘saketh’)

* o/p: SAKETH

LOWER

* it convert given values to the lower values
* ex: lower(‘SAKeth’)
* o/p: saketh

ltrim

* it remove extra spaces from the left side
* ltrim(‘ saketh’)
* o/p: ‘saketh’

rtrim

* it will remove extra spaces from the right side
* ex: rtrim (‘saketh ‘)
* o/p: ‘saketh’

trim

* it will remove the extra spaces from both sides
* ex: trim(‘ Saketh ‘)
* o/p: ‘saketh’

ex:

select \* ,length(customer\_name) 'name length',

left(customer\_name,3),

right(customer\_name,3),

lower(customer\_name),

upper(customer\_name),

rtrim(customer\_name),

ltrim(customer\_name),

trim(customer\_name)

from customers;

Stored Procedure

Stored Procedures is saves the code and reused over and over again

Normally in c/ java /python

We will write code in hellow.py like this file after writing the code we will compile and run the code run means giving inputs like 5 6

* But in stored procedure we store the queries in it and directly use where we want
* It doesn’t take much time , it increases the performance
* We don’t write query again and again

Ex:

create PROCEDURE get\_sales

AS

select sum(price) from sales1

GO;

EXEC get\_sales;

Exec get\_sales is used to excute that get\_Sales

We can also have parameterized procedures

Here we can pass parameters by using @ symbol

Like @id

And we have to pass the values to parameters

exec get\_sales 5 like this

Ex:

Create Procedure get\_sales @id INT

AS

Begin

select sum(price) from sales

where product\_ID=@id

END

exec get\_sales 5

CASTING

Casting means converting one data type to another datatype

Syntax: cast ( value as datatype)

By using this syntax we can convert any value to required type

Date : to convert to date in ‘YYYY-MM-DD’ format

DATETIME : To convert to date time like ‘YYYY-MM-DD HH:MM:SS’

Ex: select cast(2 as char);

SELECT CAST("2017-08-29" AS DATE);

DECIMAL : To convert to decimal M as maximum no of digits and D as no of digits followed by decimal point

TIME: converts to ‘HH:MM:SS’ format

CHAR :converts the value to char (with fixed length string)

NCHAR: it converts to string

SIGNED : it means it converts to signed int like -5 , +8

UNSIGNED :it converts like 6 ,7

BINARY : it converts a value to binary string

ISNULL()

It returns the specified value if the expression is null

ISNULL (null, “Saketh”)

o/p: Saketh

ISNULL(“hello” ,”Saketh”)

o/p: hello

IIF()

It returns value\_if\_true if the condition is true or else it returns value\_if\_false

Syntax:

IIF(condition, value\_if\_true, value\_if\_false)

Ex: SELECT IIF(500<1000, 'YES', 'NO')

OPERATORS

We have arithmetic operatos like +,- , \*,/

Comparision operators:

=,<=,>= ,<,> <>(not equl)

Logical operators:

AND OR NOT

AND OR

* This operator is used for the filtering purpose
* Also used for combining multiple conditions
* To test multiple conditions, we use

Conjuctive operators:

The AND and OR operators are together used for with where clause

These two operators are called conjunctive operators

Ex:

Select \* from sales

Where price>50 and sales<2

LIKE OPERATOR

It is used for searching the data

It is case insensitive

Like if search ‘apple’ it displays APPLE , Apple,aPPlE

Ex: **SELECT** \* **FROM** products **WHERE** name **LIKE BINARY** 'apple%'

**LIKE Operator Wild Cards**

Wild cards are used with the LIKE operator to search for specific patterns in strings. Wild card characters substitute one or more characters in the string. There are four wildcard characters in SQL:

1. % (Percent): Represents zero or more characters.
2. \_ (Underscore): Represents a single character.
3. [] (Square Brackets): Represents any single character within brackets.
4. – (Hyphen): Specify a range of characters inside brackets.

IN OPERATOR

Normally where clause is used to folter the data with conditions if we have more conditions means we he to write all

Like where name=’saketh’ AND NAME=’DDADJ’ AND NAME=’JDJDSK’ and name=’sakjj’

Like this

But by using in

We write directly

Where name in (‘saketh’, ‘patel’,’banu’, ‘sourabh’)

And not in

Where name NOT IN(‘saketh’, ‘patel’,’banu’, ‘sourabh’)

UNIONS

* The union is used to combine two tables but terms and conditions apply
* It provides only unique values

Terms and conditions

* Each table should have the same no. of columns
* each columns should have same data types

table 1

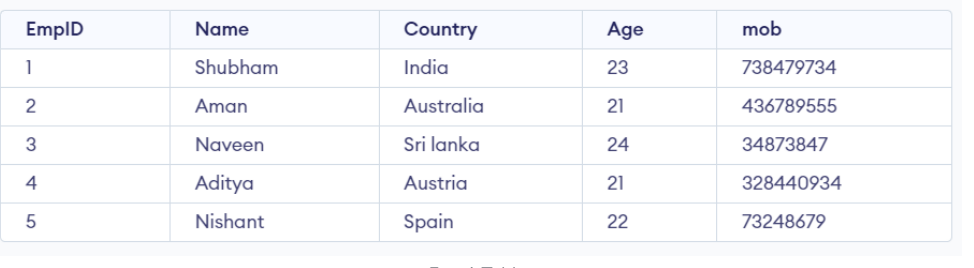
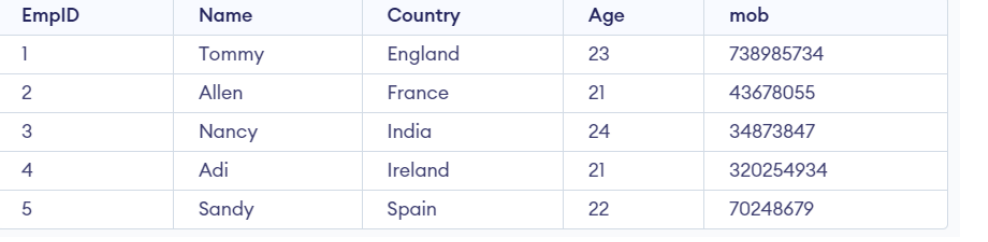
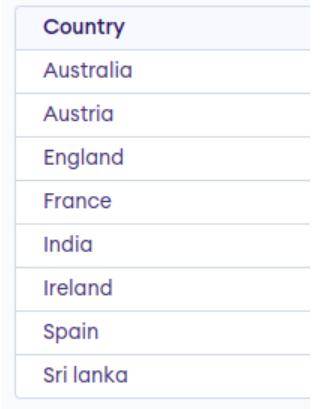


Table 2

**

Query: SELECT Country  
FROM Emp1  
  
UNION  
  
SELECT Country  
FROM Emp2  
ORDER BY Country;

o/p:



UNION ALL OPERATOR

The union all operator is different from union as the union operator removes duplicates but union all not removes

Syntax:

***SELECT****columns****FROM****table1****UNION ALL******SELECT****columns****FROM****table2;*

EXCEPT

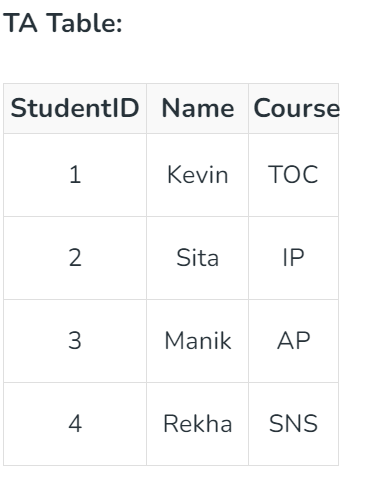
Except means it will return values from table and except some values means not return

Table 1:

Students table



Table 2



Query:

Select name from students

Except

Select name from TA;

o/p:

rohan

mansi

megha

it will print the names in table 1 only which doesn’t have in table 2

ANY

ANY compares a value of first table with all values in table 2 if there is any match it will return the row

Syntax:

Select column from table1

Where column operators ANY (select column from table2);

Ex:

SELECT \*

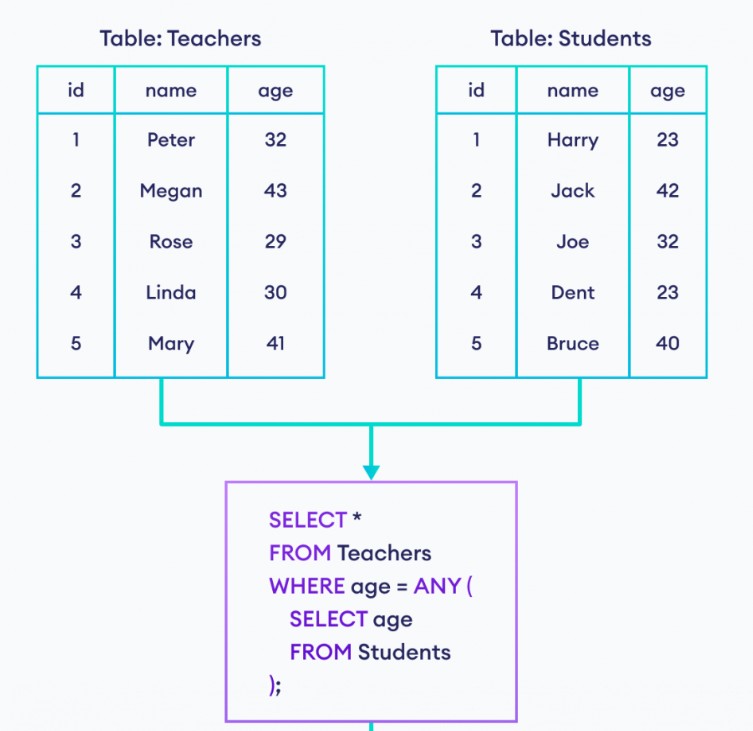
FROM Teachers

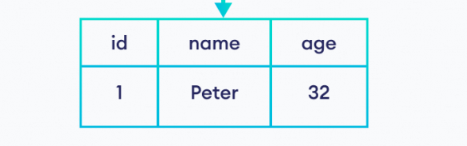
WHERE age = ANY (

SELECT age

FROM Students

);





## Example 2: SQL ANY With the < Operator

Ex: SELECT \*

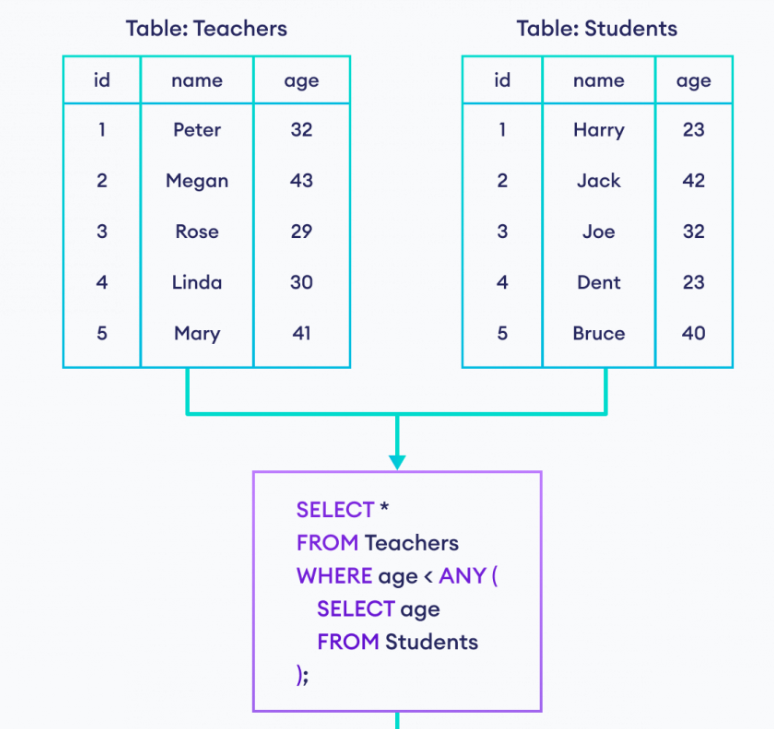
FROM Teachers

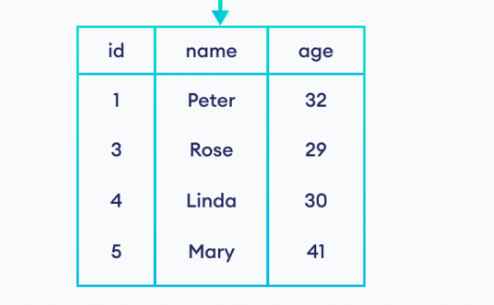
WHERE age < ANY (

SELECT age

FROM Students

);





## SQL ALL Operator

ALL operators compares a value of first table with the all values in second table and returns a row if there is match with all values

Syntax:

SELECT column

FROM table1

WHERE column OPERATOR ALL (

SELECT column

FROM table2

);

Ex: SELECT \*

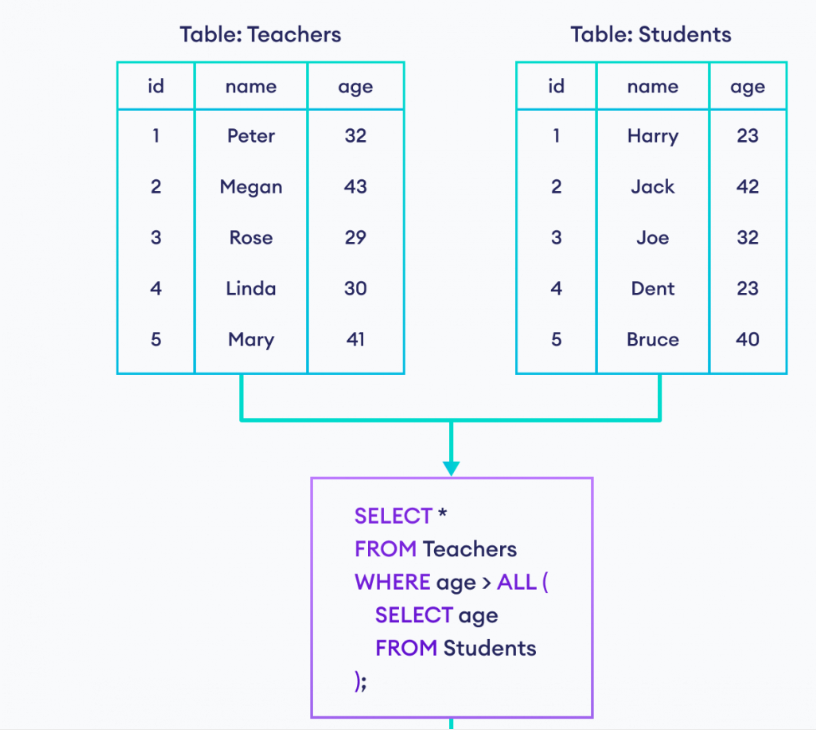
FROM Teachers

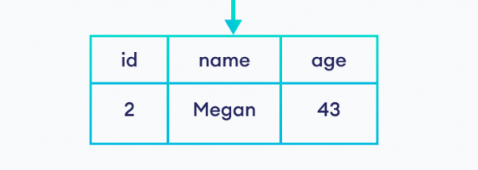
WHERE age > ALL (

SELECT age

FROM Students

);





EXISTS

It tests for existence of any record in subquery

The exists operator returns true if the subquery returns one or more records

Ex:

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20)

It will print the values

VIEWS

It will create a virtual table by selecting required columns fron tables

We can also create view by selecting columns from different tables

But it doesn’t store the data in disk like table

Ex:

**CREATE VIEW** DetailsView **AS**

**SELECT** NAME, ADDRESS

**FROM** StudentDetails

**WHERE** S\_ID < 5;

WINDOW FUNCTIONS

* normally by using aggregate functions like min, max, sum for specific operation but by using this functions we can only display the required columns
* it doesn’t display all columns in table so we want to display all columns then we use window functions

syntax:

select \*,max(salary) over() as max\_salary;

here we can get all columns as we as max\_salary column

we can also divide by groups by using

partitons

syntax:

select \* , max(salary) over(partition by dept \_name) as max\_salary

we have

* over()
* row\_number()
* rank()
* dense\_rank()
* lag()
* lead()
* ntile()

over is used to display all columns in table

ROW\_NUMBER()

It assign the row number for every row

It will assign value to each row even it is duplicate value also

RANK()

The rank function will allocate rank depends value we provided in over function

It will assign same rank for duplicate values like

1

1

And it skip 2

Then assign 3 for next one

DENSE RANK()

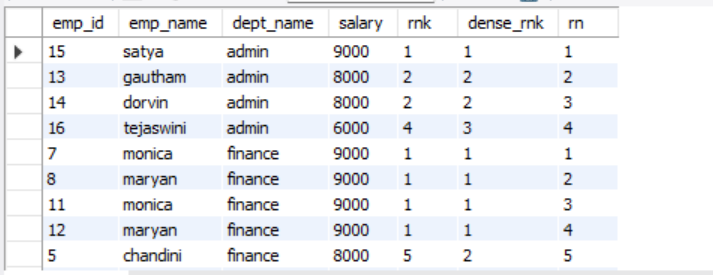
The dense rank allocates the rank

Like

1

1

For next rank as 2 it doesn’t skip value like rank function



Like this rnk means rank

Rn means row number

LAG()

The lag is assign the previous emp details to curret employee

LEAD()

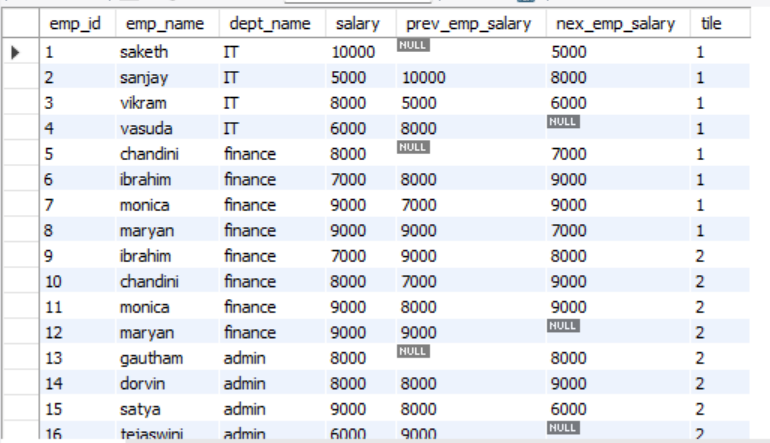
It will assign the next emp details to current employee

NTILE()

Syntax: ntile(n)

Ntile means it will divide the given rows into n number of groups

Like we give 2 means it divides the total rows as 2 groups



OPERATORS IN SQL

* arithmetic operators
* comparision operators
* logical operators
* IN
* BETWEEN
* IS NULL
* EXISTS
* ANY
* ALL
* LIKE
* UNION
* UNION ALL
* EXCEPT
* NOT
* NOT EQUAL

Aggregate functions in sql

* COUNT
* SUM
* AVG
* MIN
* MAX

DDL COMMANDS

(Data Definition language )

DDL commands in sql used to define modify and manage structure of database objects

* CREATE
* ALTER
* DROP
* TRUNCATE
* RENAME

DML COMMANDS

(Data Manipulation Language)

Dml commands used to interact with the data within database objects ,they are used to insert ,update ,delete

* INSERT
* UPDATE
* DELETE
* MERGE

MERGE

It is used to perform insert, update , or delete operations based on the results of a

CTE

* Common table expression
* Its like a temporary table that you can use in a query
* You can define it at the beginning of sql query and reference it later

If the queries have more complex in that scenario it will helpful

Easy to understand and it divides the query like function wise

We can reuse the subquery

Example: to find the average salary of employees in each department

With DepartmentAverage AS (

Select department ,avg(salary) as avg\_salary

From employees

Group by department

)

Select employees.name , employees. Salary, DepartmentAverage.avg\_salary from employees

Join DepartmentAverage on employees. Department = DepartmentAverage . department ;

We can also write directly by using group by function

But in some complex queries we use it

For counting sum

Question:

Show the provinces that has more patients identified as 'M' than 'F'. Must only show full province\_name

Tables:

Ans:

SELECT pr.province\_name

FROM patients AS pa

JOIN province\_names AS pr ON pa.province\_id = pr.province\_id

GROUP BY pr.province\_name

HAVING

SUM(gender = 'M') > SUM(gender = 'F');

o/p:

|  |
| --- |
| Alberta |
| British Columbia |
| Manitoba |
| Newfoundland and Labrador |
| Nova Scotia |
| Ontario |
| Saskatchewan |