**SQL SERVER BUILT IN FUNCTIONS**

Microsoft is providing a set of predefined SQL functions. Each function is useful to perform one task.

**NUMERIC FUNCTIONS [ GROUP Category]**

These functions are executing on columns only.

SUM(col), AVG(col), MAX(col), MIN(col), COUNT(col), COUNT(\*)

i) SUM(colname)

It will display addition of values from given column.

Ex: display addition of all salaries?

select sum(sal) from emp;

Ex: Find the sum of salaries for managers?

Ex: Find the sum of salaries of emps working under RESEARCH department?

Ex: Find the total investment amount for the products from sony?

Ex: Find the total balance from the account types SB and SAL?

ii) AVG(colname)

It will display average value from given column.

Ex: display average product cost?

select avg(cost) from prod\_dtls;

iii) MAX(colname)

It will Display highest value from given column.

Ex: Display highest salary among all salesman?

select max( sal ) from emp where job='SALESMAN';

iv) MIN(colname)

It will Display least value from the given column.

Ex: Find out least cost among all laptops?

select min(cost) from prod\_dtls where prod\_name='LAPTOP';

v) COUNT(colname)

It will Display count of NOT NULL values from given column.

Ex: find how many number of emps getting commission?

select count(empno) from emp where comm is not null;

OR

select count(comm) from emp;

vi) COUNT(\*)

It will Display number of records from given table.

Ex: display number of transactions on current day?

select count(\*) from trans\_dtls where trans\_date=sysdate;

Ex: Display number of emps under deptno 10?

select count(\*) from emp where deptno=10;

***GROUP BY clause:***

Group by clause is useful to find out aggregations based on distinct values of a column.

"The column names from select list, should be present after GROUP BY clause."

Note:

Don't use GROUP BY and DISTINCT clauses together in a query.

SYNTAX:

\*\*\*\*\*\*\*

select colname, colname,...,

aggregate1, aggregate2,.....

from table

WHERE <cond>

GROUP BY <col1>,<col2>,...

HAVING aggfunc1,...,

ORDER BY col1,col2,.....;

Ex: find out number of emps working under each dept ?

select deptno, count(\*) from emp GROUP BY deptno;

output: deptno No. of emps

--------- -------------

30 6

20 7

10 5

***HAVING clause:***

we can specify conditions on aggregate functions. By this, group by output is filtered.

Note: Without group by clause, don't use HAVING clause.

Ex: find out number of emps working under each dept if a dept has atleast 10 emps?

select deptno, count(\*) " No. of emps"

from emp

GROUP BY deptno

HAVING count(\*)>=10

order by deptno;

output: 20 70

30 60

Examples:

🡪display citynames and number of customers from each city?

🡪display cityname,gender, and number of customers from each gender from each city?

🡪Find out number of products,total investment from each comp code?

🡪find out number of accounts , min account balance, max account balance and total account balance from each account type if an act\_type has min 2 customers?

**HOW TO FIND Nth HIGHEST SALARY?**

**select distinct salary from**

**(select salary, dense\_rank() over(order by salary desc) as drk from emps) a**

**where a.drk=1;**

**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.

Ex: Get top 3 records from customers table

select top 3 \* from customers;

Ex: Get 50% records from customers table

select top 50 percent \* from customers;

**FUNCTIONS**

**ASCII**

select ascii('a')

**LEFT**

select left('hello',2)

**RIGHT**

select right('hello',2)

**LEN**

select LEN('welcome')

**LOWER**

select lower('HAI')

**UPPER**

select upper('welcome')

**LTRIM**

SELECT LTRIM(' welcome to oracle '), upper(' sql server ')

**RTRIM**

SELECT rTRIM(' welcome to oracle '), upper(' sql server ')

**REPLACE**

select replace(' welcome to oracle','oracle','sql server')

**SUBSTRING**

select substring('welcome',3,3)

**Numeric/Math Functions**

**ABS**

select abs(-9)

**CEILING**

select ceiling(9.75)

**FLOOR**

select floor(9.75)

**ROUND**

select round(123.456,1)

**Date/Time Functions**

**GETDATE**

SELECT GETDATE();

**2022-01-24 19:37:56.457**

**DATEADD**

**DATEADD( interval, number, date )**

SELECT DATEADD(YEAR,1,'2022/01/20')

SELECT DATEADD(MONTH,1,'2022/01/20')

SELECT DATEADD(DAY,1,'2022/01/20')

**DATEDIFF**

In SQL Server (Transact-SQL), the DATEDIFF function returns the difference between two date values, based on the interval specified.

**DATEDIFF( interval, date1, date2 )**

**SELECT DATEDIFF(year, '2012/04/28', '2014/04/28');**

**Result: 2**

**SELECT DATEDIFF(yyyy, '2012/04/28', '2014/04/28');**

**Result: 2**

**SELECT DATEDIFF(yy, '2012/04/28', '2014/04/28');**

**Result: 2**

**SELECT DATEDIFF(month, '2014/01/01', '2014/04/28');**

**Result: 3**

**SELECT DATEDIFF(day, '2014/01/01', '2014/04/28');**

**Result: 117**

**SELECT DATEDIFF(hour, '2014/04/28 08:00', '2014/04/28 10:45');**

**Result: 2**

**SELECT DATEDIFF(minute, '2014/04/28 08:00', '2014/04/28 10:45');**

**Result: 165**

**DATEPART**

**DATEPART( interval, date )**

**In SQL Server (Transact-SQL), the DATEPART function returns a specified part of a given date, as an integer value.**

**SELECT DATEPART(year, '2014/04/28');**

**Result: 2014**

**SELECT DATEPART(yyyy, '2014/04/28');**

**Result: 2014**

**SELECT DATEPART(yy, '2014/04/28');**

**Result: 2014**

**SELECT DATEPART(month, '2014/04/28');**

**Result: 4**

**SELECT DATEPART(day, '2014/04/28');**

**Result: 28**

**SELECT DATEPART(quarter, '2014/04/28');**

**Result: 2**

**SELECT DATEPART(hour, '2014/04/28 09:49');**

**Result: 9**

**SELECT DATEPART(minute, '2014/04/28 09:49');**

**Result: 49**

**SELECT DATEPART(second, '2014/04/28 09:49:12');**

**Result: 12**

**SELECT DATEPART(millisecond, '2014/04/28 09:49:12.726');**

**Result: 726**

SELECT DATEPART(second, GETDATE());

**DAY**

**In SQL Server (Transact-SQL), the DAY function returns the day of the month (a number from 1 to 31) given a date value.**

**DAY( date\_value )**

**SELECT DAY('2014/04/28');**

**Result: 28**

**SELECT DAY('2014/03/31 10:05');**

**Result: 31**

**SELECT DAY('2014/04/01 10:05:18.621');**

**Result: 1**

**MONTH**

**In SQL Server (Transact-SQL), the MONTH function returns the month (a number from 1 to 12) given a date value.**

**MONTH( date\_value )**

**SELECT MONTH('2014/04/28');**

**Result: 4**

**SELECT MONTH('2014/03/31 10:05');**

**Result: 3**

**SELECT MONTH('2014/12/01 10:05:18.621');**

**Result: 12**

**YEAR**

**In SQL Server (Transact-SQL), the YEAR function returns a four-digit year (as a number) given a date value.**

**Syntax**

**YEAR( date\_value )**

**SELECT YEAR('2014/04/28');**

**Result: 2014**

**SELECT YEAR('2013/03/31 10:05');**

**Result: 2013**

**SELECT YEAR('2015/12/01 10:05:18.621');**

**Result: 2015**

**@@VERSION**

**In SQL Server (Transact-SQL), the @@VERSION function returns the version of the SQL Server database, including the system and build information.**

**SELECT @@VERSION;**

**RANK()**

**DENSE\_RANK()**

**ROW\_NUMBER()**

create table emps

(

empid int,

ename varchar(10),

salary money,

desg varchar(10),

joindt date

);

insert into emps values

(1001,'harini',45000,'developer','2020/03/10'),

(1004,'ajay',40000,'developer','2010/03/20'),

(1002,'jaya',42000,'developer','2019/02/22'),

(1010,'anand',50000,'developer','2020/10/11'),

(1005,'teja',45000,'developer','2020/04/24'),

(1008,'archana',55000,'manager','2020/12/01'),

(1020,'krishna',30000,'admin','2021/08/19'),

(1006,'ishani',30000,'clerk','2020/11/11'),

(1007,'shanthi',25000,'clerk','2021/10/14'),

(1019,'siva',20000,'clerk',null),

(null,'vahini',55000,'admin',null)

go

select ename, salary, rank() over(order by salary desc) as rk,

dense\_rank() over(order by salary desc) as drk,

row\_number() over(order by salary desc) as rn

from emps;

sample output

ename salary rk drk rn

archana 55000.00 1 1 1

vahini 55000.00 1 1 2

anand 50000.00 3 2 3

teja 45000.00 4 3 4

harini 45000.00 4 3 5

jaya 42000.00 6 4 6

ajay 40000.00 7 5 7

krishna 30000.00 8 6 8

ishani 30000.00 8 6 9

shanthi 25000.00 10 7 10

siva 20000.00 11 8 11

**HOW TO DELETE DUPLICATE RECORDS?**

create table depts

(

did int,

dname varchar(20),

address varchar(20)

);

insert into depts values

(111,'production','hyd'),

(111,'production','hyd'),

(111,'production','hyd'),

(222,'sales','india'),

(222,'sales','india'),

(222,'sales','india')

go

with deptcte as

( select \*, row\_number()

over(partition by did,dname,address order by did,dname,address) as rn

from depts)

delete from deptcte where rn>1;

select \* from depts;

Common Table Expression

CTE is common table expression. It is a temporary named result set derived from a simple query and defined within the execution scope of a single SELECT, INSERT, UPDATE, or DELETE statement. CTE can be used for writing complex recursive queries. It is much more popular than temporary tables.