**CONTENT**

**SL NO Content Page NO**

Sql Server commands 2

Constraints 4

Data Types 7-12

Windows Function

1 Set Operator 17

2 CTE 18

2 Index 19-23

3 Views 23-25

**Performance tuning**

* Count(1) instead of count(\*)
* **Set nocount on** – it won’t count the rows like rows affected.
* Tables should be in **De-Normalized form**
* **Partitioning the tables** if tables has more row sit would take more time to fetch, to enhance query performance by limiting the amount of data scanned for a specific query.
* Retrieve only the **necessary columns** instead of using SELECT \*
* Analysing Execution Plans
* Use Stored Procedures
* **Optimize Joins**: Choose appropriate join types ( INNER JOIN, LEFT JOIN) based on the relationship between tables.
* **While writing query alias name in select statement we can directly use it only for order by clause, it will show invalid column name for grp by and having clause.**

**SQL SERVER**

**DataBase**

**To create database**

Create database database\_name

Create database sample1

**TO alter the DB name**

Alter database old database\_name modify name=new database\_name

Alter database sample1 modifyname= sample2

**TO drop the DB name**

Drop database database\_name

drop database sample2

**TO change database**

use [databasename]

write your query

go

TO find the no\_of database

select \* from sys.databases

TO find the no\_of tables in database

select \* from sys.tables

select name from sys.tables

select \* from sys.databases

select \* from sys.views

select \* from sys.procedures

select \* from sys.tables

select \* from INFORMATION\_SCHEMA.COLUMNS

select \* from INFORMATION\_SCHEMA.tables

**Creating schema to the database**

For every database we have to create new schema

Schema is a list of logical structures of data.

DBO database owner,

create schema abc -- to create schema,

Drop schema abc -- to drop schema must and should no data inside it

**Creating table to the database**

**Create table table\_name(column \_name dataType(size) constriants)**

Create table demo11( id int notnull unique, name nvarchar(20)notnull, age int notnull, DOB date notnull)

**Rename table name**

sp\_rename'DEMO11','demo'

Note – **“ if we change table name it will affect stored procedure which depends on it”**

**DROP table name**

Drop table demo

--------------o-r---------------------------

IF OBJECT\_ID ('new\_employees', 'U') IS NOT NULL --U = Table (user-defined)

DROP TABLE new\_employees;

**Truncate table name**

truncate table demo

**drop column**

alter table tablename drop column columnname

**Add column**

alter table tablename add columnname datatypr constraints

**Rename column name**

sp\_rename'table\_name.old\_column\_name','new\_column\_name'

sp\_rename'DEMO.age','ages'

**to add constraints for existing table**

alter table demo\_table add primary key (id)

**create table from another table and values also**

select \* into new\_table name from old\_table name

**create table from another table if given condition is false it will create only tables.**

select \* into new\_table name from old\_table name

where 1=11;

**Inserting the values to the Table**

Insert the values

insert into demo values(1,'pradeep',25,'1998-02-20');

--begin

--insert into demo values(2,'raama',11000,'1899-02-20');

--insert into demo values(3,'seeta',500,'1899-02-20');

--end

SELECT \* INTO DUMMY FROM SHA

Retrieve the values from table

Select \* from table\_name

--select \* from demo;

Update the existing values

--update demo set age=1000

--where id=3;

Deleting the values from table

--delete from demo

--where name='pradeep'

1. Update the patients table for the allergies column. If the patient's allergies is null then replace it with 'NKA'.

UPDATE patients

SET allergies = 'NKA'

WHERE allergies IS NULL;

**Inserting the values to the exisiting Table from table**

select \* into abc.demo from orders ------to copy the data by existing table into creating new table

insert into abc.demo select \* from orders ---to insert data from one existing table to another existing table

**Synonym** – Alternative name given to the table.

Create synonym h for human

Tuple is known as rows.

Attributes is known as column.

**CONSTRAINTS**

Constraints given to the column for validating the data.

When we insert data to a column, constraints will validate data before entering into the column. Set of rules given to the column

* **NOT NULL** : This constraint won’t allow to store a null value in a column. We must and should fill values.
* **UNIQUE**: All the values in the column must be unique. That is, the values in any row of a column must not be repeated, which accepts one null value. we can have multiple unique constraints.
* **PRIMARY KEY**: A primary key which can uniquely identify each row in a table. We can have only one primary key in a table, which does not accept null & duplicate value.
* **FOREIGN KEY**: it is used to build relationship between two tables. It can accept multiple null values, and duplicate values, we can have multiple foreign key.
* **CHECK**: This constraint helps to validate the data of a column to satisfies a particular condition. age int check(age >18)
* **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.
* create table t1(id int, nam varchar(10), dob int default 100)
* insert into t1(id,nam) values(1,'pews');

if we mention only inserting column name then only default will work

First Parent table we have to create then child table.

What ever the data given in primary key, same should be added in FK if extra data is added it will through error like.

**The INSERT statement conflicted with the FOREIGN KEY constraint "FK\_\_b\_\_id2\_\_4BAC3F29". The conflict occurred in database "class", table "dbo.a", column 'id'.**

the 2 table which has primary and foreign key relation ship, we can’t delete any records from parent table, we can delete from child table.

create table s1 (id int primary key , name varchar(20), age int )

create table s2 (ids int foreign key references s1(id), name varchar(20), age int )

**Composite primary key** **–** A Composite Primary Key is created by combining two or more columns in a table that can be used to uniquely identify each row in the table.

Values in a individual column can be duplicate but across column must be unique.

If we add same value it will through error   
**Violation of PRIMARY KEY constraint 'z'. Cannot insert duplicate key in object 'dbo.c'. The duplicate key value is (1, 2).**

RDBMS won’t allow to built a many to many relation ship between 2 tables by using junction table.

create table s1 (id int , ids int constraint pk primary key(id,ids ))

**IDENTITY**

Identity will automatically generate a **unique and incremental** value for a column.

Identity columns are com monly used for **primary keys** to ensure uniqueness of each record in a table.

**SYNTAX**

column name datatype identity(1,1) starting value is 1 incremental value is 1

**set identity\_insert tablename ON** = it is useful when you need to insert specific values that are not generated automatically by the identity column.

**set identity\_insert tablename off** = to turn off identity property u can use this code.

**select scope\_identity()** - is used to retrieve last generated identity column value under the same session and same scope.

**select @@identity** - is also used to retrieve last generated identity column value under the same session across any scope

**select IDENT\_CURRENT('tablename')** - specific table across any session and any scope.

create table gauvrav (id int identity(1,1), name varchar(20))

insert into gauvrav(id,name) values(1000,'bora')

select \* from gauvrav

set identity\_insert gauvrav on

set identity\_insert gauvrav off

DBCC checkident (EMPidentity, RESEED, 20)

**CLAUSE**

Clause in SQL is a built-in function that is used to retrieve the data from the records present in the database.

**ORDER BY clause**

The ORDER BY statement in SQL is used to sort the fetched data in either ascending or descending according to one or more columns.

* By default ORDER BY sorts the data in ascending order.
* We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

**IN:** It is multi valued operator.

The IN operator allows you to specify multiple values in a WHERE clause.

0

**LIKE :**

It is mainly uses for pattern matching the LIKE operator is mainly used in the WHERE clause to search for a enumerate pattern in a column.

1. %: Used to match zero or more characters. (Variable Length)
2. \_: Used to match exactly one character. (Fixed Length)

**IS:**

It is used for comparing with null values and not null.

**Between:**

The Between operator selects values within a given range.

Syntax

Column\_name between(Lower\_range and Higher\_range).

**Select Statement**

Select clause is used to display the data from the table and selects the column.

select clause uses the distinct clause to remove the duplicates rows.

**Where Clause**

It will filter the records from the table, it will execute before group by clause, we can use DML operations.

select p\_name from product where p\_name like'[mw]%'

-------------------or-----------------------

select p\_name from product where (p\_name like'm%'or p\_name like'w%')

**Top clause**

It will select the top of no: of list as mentioned

Select top 2 \*from product

it will select TOP 2 records from table

select top 50 percent \* from product

it will select TOP 50% of records from table

**Group by clause**

It will groups the records.

if we use any column besides the aggregate function or multi row function in select clause, we must and should use the group by that column name;

**Where Clause**– it will filter the records from table,it will execute first.

**Having Clause**- it will filter the records from group by clause, it will execute after group by, and also perform some aggregation task.

**Alias :**

Temporary name given to the result set of column.

* An alias is created with the AS keyword.

SELECT column\_name AS alias\_name  
FROM table\_name;

**SQL query order of execution**

FROM clause

ON clause

OUTER clause

WHERE clause

GROUP BY clause

HAVING clause

SELECT clause

DISTINCT clause

ORDER BY clause

TOP clause

select count(\*) from employees -- counts rows including null

select count(10) from employees -- counts the rows of column 10 including null

select count(manager\_id) from employees -- counts particular mentioned column rows

without null

select sum(1) from employees -- sums the number of rows in first column

select sum(10) from employees -- sums the number of rows from 1 to 10 column

**DATATYPES**

What type of data to be stored inside column.

Int,big int, small int

Date

Char, nChar

Varchar, nVarchar

**CHARACTER FUNCTION**

[ASCII](https://www.w3schools.com/mysql/func_mysql_ascii.asp) Returns the ASCII value for the specific character

selectASCII('A')--65 to 90

selectASCII('b')--98 --97 to 122

selectASCII('0')--48-57

CHAR Returns the character for the ASCII value.

selectchar(65)--A

selectchar(100)--d

declare @start int

set @start = 65

while(@start<=90)

begin

printchar( @start)

set @start= @start+1

end

[CONCAT](https://www.w3schools.com/mysql/func_mysql_concat.asp)Adds two or more expressions together.

select(CONCAT('p','rad','e','EP'))

select'p'+'R'+'adee'+'P'

[CONCAT\_WS](https://www.w3schools.com/mysql/func_mysql_concat_ws.asp) Adds two or more expressions together with a separator

selectCONCAT\_WS('-','today','is','monday')today-is-monday

[LTRIM](https://www.w3schools.com/mysql/func_mysql_ltrim.asp)Removes spaces from a string from left side

selectlen(' 12345678')--10

selectlen(LTRIM(' 12345678')) --8

[RTRIM](https://www.w3schools.com/mysql/func_mysql_ltrim.asp)Removes spaces from a string from right side

selectlen('12345678 ')--10

selectlen(RTRIM('12345678 ')) –8

Trim() Removes leading and trailing spaces from a string only at end

selecttrim('#%'from'#pradeep%') --pradeep

selecttrim('#%'from'#prad#e%ep%') --prad#e%ep

Upper() Converts a string to upper-case

selectupper((CONCAT('p','rad','e','EP')))

Lower() Converts a string to upper-case

selectlower('PraDEEP') -- pradeep

Len() gives the length of the string

selectlen('prade2-ep') --9

Replace() Replaces all occurrences of a substring within a string, with a new substring

selectREPLACE('pradeep','a','z')--przdeep

Replicate() which repeats the task multiple no of times

selectREPLICATE('jai sri raam ',108)as slogan

Reverse() Reverse a string and returns the result

selectreverse('mara')--aram

STUFF() replace the particular char

select STUFF('ABCDEFGHIJKLMNOP', 3,5,'XYZ')--ABXYZHIJKLMNOP

Substring() Extracts a substring from a string

SELECTSUBSTRING('PRADEEP',5,2) --EP

UNICODE() provides the Unicode values of given character

SELECTUNICODE(‘p’) --115

TRANSLATE() replaces a of characters in a string with another sequence of characters

selecttranslate('Monday','mon','sun')--sunday

Trim() Removes the initial mentioned chars and spaces from a string

selecttrim('.@-1'from'.--1@pradeep@1--.');--pradeep

CHARINDEX() gives the index position of given char in a string

selectcharindex('@',pk@gmail.com) --3

--it will also take 3 args, 3rd arg is the starting position of the index

selectCHARINDEX('a','pradeaep',4)-----6

PATINDEX() gives the index position number when the pattern matches string and it should be in sequential order. It has only 2 arg

% is mandatory

selectPATINDEX('%ef%','abcdef') --5

selectPATINDEX('%abc%','abcdef') --1

selectPATINDEX('%abdc%','abcdef') --0

selectPATINDEX('%efg%','abcdef') --0

[RIGHT](https://www.w3schools.com/mysql/func_mysql_right.asp) () Extracts a number of characters from a string (starting from right)

selectright('ABCDEFGHIJKL',4) --IJKL

Left() Extracts a number of characters from a string (starting from left)

selectLEFT('ABCDEFGHIJKL',3) --ABC

DataLength() – it will consider the length of the string including spaces

selectDATALENGTH(' pradeep ') ---12

PARSENAME()-- it will fetch from right side after dot .

select PARSENAME('abc,def,ggd.gd,ghi',1) as a

declare @mystring varchar(100)=' hello hi everyone '

select @mystring +'has'+' '+' '+cast(datalength(@mystring) datalength(replace(@mystring,' ',''))asvarchar)+' '+'Spaces'

**COALESCE():**COALESCE function in SQL returns the first non-NULL expression among its arguments. If all the expressions evaluate to null, then the COALESCE function will return null.

**It will accepet multiple arg**

**ISNULL() :**

The SQL Server ISNULL() function replaces NULL with a specified value. And it accepts only 2 arguemnt.

isnull(Resort\_name,'no resort')

**STRING\_AGG()**

**If given condition satisfies, it will combine the records**

select id,STRING\_AGG(upper(sname),', ') from sha

group by id

SHIFT ALT DOWN ARROW

**String Split()**

**It will works opposite as string argg**

**NUMERIC FUNCTION**

Abs() Returns the positive value of a number

selectabs(-123) --123

CEILING() Returns the largest integer value that is >= to a number

selectCEILING(12.1) --13

Floor() Returns the smallest integer value that is <= to a number

selectFLOOR(12.8) --12

Power() Returns the value of a number raised to the power of another number

selectpower(2,3) --8

sqrt() Returns the square root of a number

selectsqrt(4) --2

Square() Returns the square root of a number

selectsquare(4) --16

Modulus

% Returns the remainder of a number divided by another number

select 4%3 --1

Rand() Returns a random number

selectRAND() --0.163939806910158

Round()

selectROUND(10.62343,1) --10.60000

selectROUND(10.62343,2) --10.62000

selectROUND(10.62343,3) --10.62300

selectROUND(10.62343,4) --10.62340

Cast() = it will convert the one data type to another, not in base table, only while executing the queries.

Which is ANSI (American National Standard Institute ) YYYY-MM-DD

Select cast((price-cost)as decimal)/cast(cost as decimal(10,2))\*100 from product

for percentage calculation.

decimal(10,2) 10 means before point value and 2 means decimal values.

Convert() - converts to require format of date and time

Which is not an ANSI (American National Standard Institute )

Select CONVERT(varchar,get date(),100) 100 is format style of 12hrs

select convert(int,123) ---123

Mar 10 2023 11:25AM

1st arg is date

2nd arg is months to be increase.

3rd arg is style format

**DATE FUNCTION**

getdate() – returns current date and time

select getdate() 2023-03-10 11:27:15.790

sysDateTime() - returns current date and time with more precision of **seconds**

selectSYSDATETIME()

2023-03-10 11:28:54.2598560

DateFRomParts() – returns the date value specified like year,month,date.

selectdatefromparts(YEAR(GETDATE()),1,1) --2023-01-01

1st arg is year

2nd arg is months

3rd arg is day

EOMONTH() End of the month- it will last day of the specified month.

selectEOMONTH(GETDATE(),0) --2023-03-31

1st arg is date

2nd arg is months to be increase.

DateName() – return the name of the year,month,day etc.

selectdatename(YEAR,getdate())--2023

selectdatename(month,getdate())--march

selectdatename(day,getdate())--10th day

selectdatename(WEEK,getdate())--10th week

selectdatename(WEEKDAY,getdate())--friday

selectdatename(QUARTER,getdate())--1st quarter

selectdatename(HOUR,getdate()) --11

selectdatename(MINUTE,getdate()) --35

selectdatename(SECOND,getdate())--21

1st arg is interval like year,month,day,week,weekday,hour,minute,second

2nd arg is date.

DatePart() – returns the single part of date.

selectdatepart(YEAR,getdate())--2023

selectdatepart(month,getdate())--3

selectdatepart(day,getdate())--10th day

selectdatepart(WEEK,getdate())--10th week

selectdatepart(WEEKDAY,getdate())--6

selectdatepart(QUARTER,getdate())--1st quarter

selectdatepart(HOUR,getdate())--11

selectdatepart(MINUTE,getdate())--49 min

selectdatepart(SECOND,getdate())--55

1st arg is interval

2nd arg is date

DateDiff() - returns the difference between two dates

selectDATEDIFF(year,'1998-02-20',GETDATE());--25

selectDATEDIFF(MONTH,'1998-02-20',GETDATE());--301

selectDATEDIFF(WEEK,'1998-02-20',GETDATE());--1307

selectDATEDIFF(DAY,'1998-02-20',GETDATE());--9149

selectDATEDIFF(HOUR,'1998-02-20',GETDATE());--219587

1st arg is interval

2nd arg is from date

3rd arg is to date

DateAdd() – adds or substracts the specified interval of time

selectGETDATE() --2023-03-10 11:49:40.560

selectDATEADD(year, 1,GETDATE());--2024-03-10 11:49:40.560

selectDATEADD(month, 1,GETDATE());--2023-04-10 11:49:40.560

selectDATEADD(month,-3,GETDATE());--2022-12-10 11:52:28.493

selectDATEADD(day, 1,GETDATE());--2023-03-11 11:49:40.560

selectDATEADD(HOUR, 1,GETDATE());--2023-03-10 12:49:40.560

1st arg is interval

2nd arg is adding value

3rd arg is to date

Day() -- returns only day

selectday(getdate()) ---return day in int in 30days

Moth() – returns only month

selectmonth(GETDATE())--return month in int in 12 months

Year() --return only year

selectyear(GETDATE())--return year in int in 2023

isDate() – checks the given date is valid or not if it true returns 1 , not true return 0

selectISDATE(1998-02-20) --1

selectISDATE('2023-02-30') --0

DateTrunc() – it will removes the timing

SELECTDATETRUNC(WEEK,GETDATE()) --2023-03-12 00:00:00.000

Format( ) – it will return the result as mentioned

selectformat(getdate(),'ddd d mm yyyy')---Wed 15 21 2023

**dayofyear** – it will count the day from first day of this year to till date

select datepart(dayofyear,(getdate()))

find te date

--2) to finf the next month

selectdatename(month,DATEADD(month,1,getdate()))

--------------------- or---------------------

selectmonth(GETDATE())+1

1).what Is the age of the account

select pat\_id,doc\_id,admission\_date,datediff(year,admission\_date,getdate())from casetable

2) who opended account current year

select pat\_id,doc\_id,admission\_date,datediff(year,admission\_date,getdate())from casetable where datediff(year,admission\_date,getdate())=0

3) how many customers opened the account last 3 year

selectcount(\*),year(admission\_date)as [year] from casetable

wheredatediff(year,admission\_date,year(2018-05-12))<3

groupbyyear(admission\_date)

**Window Function**

It will create seperate window frame for group of records

* AGGREATE window Function
* Ranking Window Functions
* Analytical Window Function

**AGGREATE or MULTI ROW FUNCTION**

**Takes multiple input gives single output**

* **MAX()** – It will returns maximum value
* **MIN()**– It will returns minimum value
* **AVG()**– It will returns Average value
* **SUM()**– It will returns Sum value
* **COUNT()**– It will returns Count value

select \*,

sum(sal) over(partition by grade order by id) as sl,

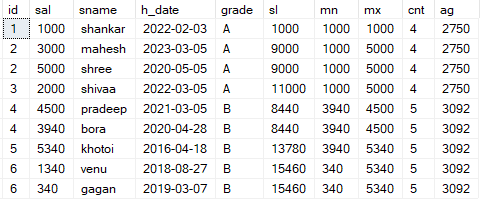
min(sal) over(partition by grade order by id) as mn,

max(sal) over(partition by grade order by id) as mx,

count(sal) over(partition by grade) as cnt,

avg(sal) over(partition by grade) as ag

from sha

****

**Ranking Window Functions:**

These functions ranks each row of a partition in a table.

**RANK()** -It's used to **generate a unique rank** for each row in a table based on the specified value. If this function gets the two records with the same value, it will assign the same rank to both records and skip the next ranking.

**DENSE\_RANK()** - It works the same as the RANK() function except that it does not skip any rank. It always assigns rank in consecutive order. It means that when two records are found equal, this function will assign the same rank to both records and the next rank being the next sequential number.

**ROW\_NUMBER()** - It is used to assign a unique sequential number to each record within the partition. It always starts with one and increases by one until all the records in a partition are not reached.  the row\_number function doesn’t assign same rank where there are duplicate values for the column

**NTILE()** -This window function **distributes rows into a pre-defined number (N) of approximately equal groups**. Each row group is assigned a rank depending on the defined condition, and the numbering begins with the first group. It enables us to determine which percentile (or quartile, or other subdivision) a particular row belongs to.

* **The partition\_by\_clause** divides the result set produced by the FROM clause into partitions to which the function is applied. If not specified, the function treats all rows of the query result set as a single group.
* **The order\_by\_clause** determines the logical order in which the operation is performed. The order\_by\_clause is required.

# OVER Clause () :  defines a window or user-specified set of rows within a query result set. Determines the partitioning and ordering of a rowset before the associated window function is applied. That is, the OVER clause defines a window or user-specified set of rows within a query result set.

OVER (

[ <PARTITION BY clause> ]

[ <ORDER BY clause> ]

[ <ROW or RANGE clause> ]

)

RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

select id,sal,sname,h\_date,grade,

rank() over(partition by grade order by id) as rnk,

DENSE\_RANK() over(partition by grade order by id) as Ds\_rnk,

ROW\_NUMBER() over(partition by grade order by id) as rw\_nmbr,

ntile(2) over(partition by grade order by id) as tile

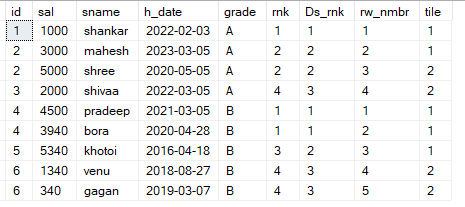
from sha

inserting data into the temp table and fething from it

select \*, DENSE\_RANK() over( order by sal)as n into #abc from emp

select \* from #abc

where n=3



**Analytical Function**

**Value Window Functions:** These functions are locally represented by a power series.

* **LAG()-** it will access the previous row data and applied to current row.
* **LEAD()-** it will access the next data and applied Over by clause is mandatory in that order by clause is mandatory, partition is optional.
* **FIRST\_VALUE() & LAST\_VALUE()** - These functions are used to find the first and last record in the table or a partition if the PARTITION BY clause is specified. Here we should note that these functions are mandatory to use the **ORDER BY clause**.

select \*,

FIRST\_VALUE(sal) over(partition by grade order by sal) as FV,

LAST\_VALUE(sal) over(partition by grade order by sal range between unbounded preceding and current row) as LV\_CR,

LAST\_VALUE(sal) over(partition by grade order by sal range between unbounded preceding and unbounded following) as LV\_UF,

LEAD(sal) over(partition by grade order by sal) as led,

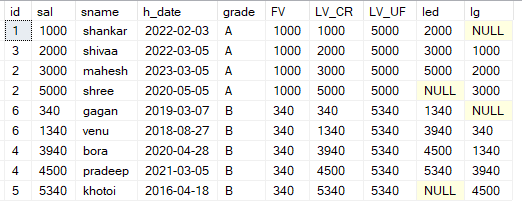
LAG(sal) over(partition by grade order by sal) as lg

from sha

LEAD(Expression , Offset , default)

LEAD(col\_name , 5 (skips 5 rows) , 0 (it will replace 0 other wise it will be null))

select \*,lead(empno,5,0) over( order by deptno) from emp

****

**SUB Queries**

**Select Statement inside the select statement is called sub query.**

**Inner query will execute first then outer query will execute,output of inner query is the input of outer query.Inner query is independent and out query is dependent on inner query.**

**sub queries can be nested upto 32 levels.**

**Types of Sub Query**

1. **Single Row Sub Query**
2. **Multi Row Sub Query**

**Single Row Sub Query**

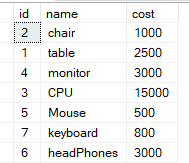
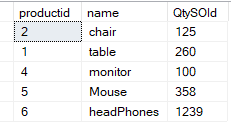
If your subquery returns only one row that is called **Single Row Sub Query**

**Multi row sub query**

If your subquery returns more than one row, it can be referred to as a **multiple-row subquery**. this subquery type includes

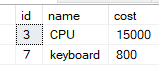
(1) subqueries that return one column with multiple rows.

(2) subqueries that return multiple columns with multiple rows.

** **

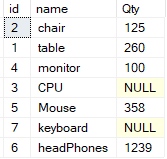
**Tbl1 Tbl2**

select \* from tblProduct where id not in(select productid from tblProductsales)



**Output**

select id, name, (select Qtysold from tblProductsales where id=productid ) as Qty from tblProduct



multi row

select \* from sha where sal < any (select sal from sha where sal>2000) ---- less than max

select \* from sha where sal > any (select sal from sha where sal>2000) ----greater than min

select \* from sha where sal = any (select sal from sha where sal>2000) --- same as IN operator

--===============

select \* from sha where sal < all (select sal from sha where sal>2000) ---- less than min

select \* from sha where sal > all (select sal from sha where sal>2000) ----greater than max

select \* from sha where sal = all (select sal from sha where sal>2000) --- same as not exixts operator

**Co-related**

 a SQL correlated subquery contains a query within a query. It gets its name because the two queries are related; the inner query uses information obtained from the outer query.

SELECT

  lastname,

  firstname,

  salary

FROM employee e1

WHERE NOT EXISTS (SELECT project\_id

                  FROM assigned\_to\_project

                  WHERE employee\_id = e1.employee\_id)

**DERIVED TABLE:**Derived tables are the tables which are created with the help of the Select statement. Derived table expression appears in the FROM clause of a query. In derived table server create and populate the table in the memory, we can directly use it and we also don’t require to drop the table. But scope of derived table is limited to the outer Select query who created it. Derived table can’t be used further outside the scope of outer select query.

1. SELECT SUM(Tab.Salary)AS Total\_Salary,Tab.Age
2. **FROM**
3. (SELECT e.Salary,e.AgeFROMdbo.Emp e
4. WHERE e.Age>20)Tab
5. GROUPBY Tab.Age

**JOIN**

Joins are used to retrieve the data from multiple table, by joining the tables.

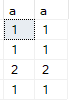
Types of joins

1. Cross join
2. Inner join
3. Left Anti join
4. Right Anti join
5. Outer join

* Left outer join
* Right outer join
* Full outer join

**Inner join**- it will retrieve the matching records between the tables.

select \* from join1 a inner join join2 b on a.a=b.a

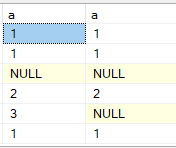
C:\Users\Admin\Desktop\t1.PNG  

T1 T2 output

**Left outer join**

It will retrieve the matching records between 2 tables and unmatched records from left table only.

select \* from join1 a left join join2 b on a.a=b.a

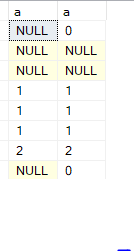
C:\Users\Admin\Desktop\t1.PNG  

T1 T2 output

**Right outer join**

It will retrieve the matching records between 2 tables and unmatched records from Right table only.

select \* from join1 a right join join2 b on a.a=b.a

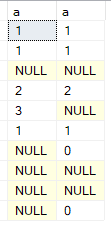
C:\Users\Admin\Desktop\t1.PNG  

T1 T2 output

**Full outer join**

It will retrieve the matching records between 2 tables and unmatched records from 2 table only.

select \* from join1 a full join join2 b on a.a=b.a

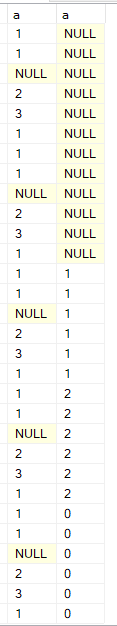
C:\Users\Admin\Desktop\t1.PNG  

T1 T2 output

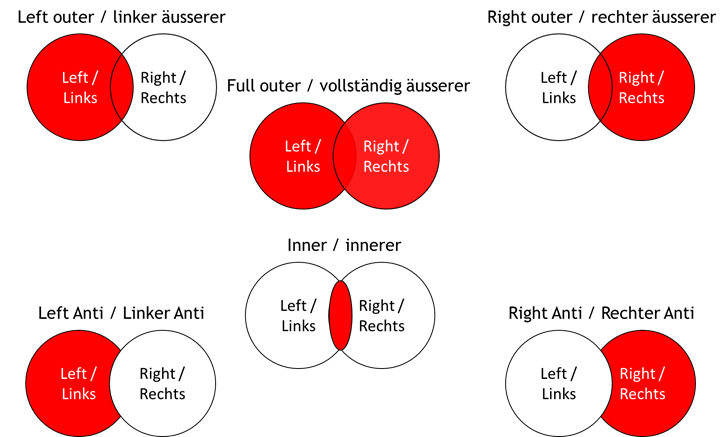
**Cross join**

When ever we perform the join b/w 2 table every row in the first table will map with all the rows in second table.

select \* from join1 a cross join join2 b

C:\Users\Admin\Desktop\t1.PNG  

T1 T2 output



**SET OPERATOR**

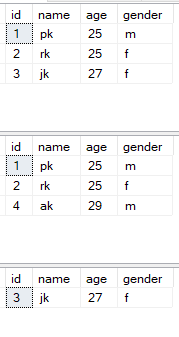
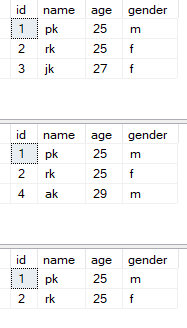
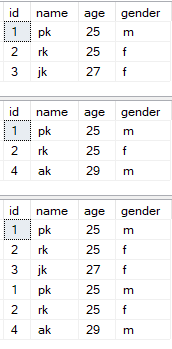
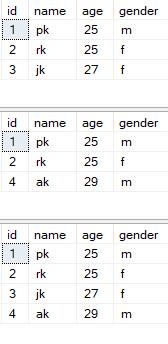
SQL set operators are used to combine the results obtained from two or more queries into a single result.

Few points which we should be kept in mind while working with SQL set operators:

* The number of columns in the SELECT statement on which we have applied SQL set operators must be the same.
* The selected columns must have the same data type.
* The order of the columns must be in the same order as mentioned in the SELECT statement.

**Types of SET OPERATOR**

1. **UNION** – it will combine all records from two tables which is not repeated, no duplicates are allowed.and sorts the data.
2. **UNION ALL**– it will combine all records from two tables and fetch everything, even duplicates are also fetched.
3. **INTERSECT :**  The intersect clause is used to provide the result of the intersection of two select statements. This implies the result contains all the rows which are common to both the SELECT statements. (Inner join )
4. **EXCEPT OR MINUS**–Uncommon values from left tables only. (left anti join)

****

**A B C D**

**Common Table Expression CTE**

* The CTE is an abbreviation of “Common Table Expression,It is a temprory result table that is created and works within the execution time of a single select, insert, update, delete statements.
* CTE exists in the memory only while the query is running. After query is run, CTE is discarded, it can’t be used in the next query.
* the scope is limited to the current query.
* CTE improves readability and ease in maintenance of complex queries and sub-queries.
* If we use sub query it will take so many time, the replace of sub-query if we use CTE the performance is far better than as compared to sub-query

We can use CTE instead of sub query,

**Syntax**

With cte\_name

As

( CTE\_Definition)

Sql Definiton

**For inserting data**

With cte1

As

(select \* from emp)

Insert cte1 values(1,’upendra’,78686283) -----**into** not used in cte

Delete cte1 where id=1 ----**-from** not used in cte.

**Recursive CTE**

A recursive CTE is a CTE that references itself. By doing this the CTE repeatedly executes, returns set of data,untill the recursive get terminated.

A recursive CTE is useful in querying hierarchical data like one employee reports to a manager  or multi-level bill of materials when a product consists of many components, and each component itself also consists of many other components.

**A recursive CTE has three parts:**

* An initial query that returns the base result set of the CTE. The initial query is called an **anchor member query**.
* A recursive query that references the common table expression, therefore, it is called the **recursive member**. The recursive member is union-ed with the anchor member using the [**UNION ALL**](https://www.sqlservertutorial.net/sql-server-basics/sql-server-union/)**operator**.
* A **termination** condition specified in the recursive member that terminates the execution of the recursive member.

**Print Monday to Saturday by using recursive**

with cte(n,[wek]) as

(select 0,DATENAME(WEEKDAY,0)

union all

select n+1,dATENAME(WEEKDAY,n+1) from ccte where n<=5 )

select \* from ccte

**print 1 to 100 from recursive cte**

with num(n)

as

(select 0

union all

select n+1 from num where n<100)

select \* from num

**print 1 to 200 from recursive cte**

with num(n)

as

(select 0

union all

select n+1 from num where n<200)

select \* from num option(maxrecursion 200)

**PIVOT**

It will groups the records as per column wise. Pivot will convert the row data into column data into more simpler.

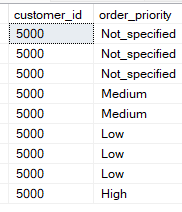
**Syntax**

Select col1\_name,col2\_name from(select \* from table) as a

Pivot (aggregate function from (col to be aggregate) in ()) as b

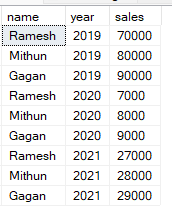
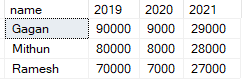
select customer\_id,not\_specified,[medium],[low],[high] from (select customer\_id, order\_priority from customer1) as a

pivot (count(order\_priority) for order\_priority in (not\_specified, [medium],[low], [high])) as b;

C:\Users\Admin\Desktop\p3.png

select name,[2019],[2020],[2021] from (select name,year,sales from demo) as a

pivot(count([year]) for [year] in ([2019],[2020],[2021])) as b

** **

**Before Pivot Pivot**

**INDEX**

##### **How is data stored in a database?**

1. SQL Server stores data under data pages(Data block smallest unit where data stored), where a data page is a memory location for storing the information.
2. A data page will be having a size of 8KB and every 8 data pages we stored under a logical container known as “extent”64kb.

**The database engine retrieves the information from the table by two different mechanisms for searching the data.**

1. **Full page scan -** SQL Server will search for the required information in each and every data page to collect the information. So, if the tables having more number of rows it will take lots of time for searching all the data so it is a time-consuming process.
2. **Index Scan -**  it will make use of an index for retrieving the information, where an index is a pointer to the information what we retrieve which can reduce the disk I/O operation saving the time.

Without index datas are stored in heap.

##### **What is an index?**

* Indexes are used to find data from tables quickly. Indexes are created on tables and views.
* we can save time and we can improve the performance of database queries and applications.
* If there is no index , then the query engine checks every row in the table from the beginning to the end. This is called a **Table Scan**. A table scan is bad for performance.
* If any column is updating frequently we have to avoid index.
* If we use a column more than one times we need index.
* When the column has large number of large null values we have to create index.
* The users cannot see the indexes, they are just used to speed up searches/queries.

**Note**: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update).

**To Create index**

Create index index\_name on table\_name (column for the index)

Create index ix\_human on human(salary asc)

**TO Drop Index**

Drop index table\_name.index\_name

dropindex human.ix\_human

index

Root node

Intermediate node

Leaf node – datas are stored in it.

**Types of Index**

1. Clustered
2. Non clustered
3. Unique
4. Filtered
5. XML
6. Full Text
7. Spatial
8. columnStore
9. Index with include column
10. Index on computer column

**CLUSTERED INDEX.**

Indexes are created inside the original table only and datas are sorted order wise in a table.

a table can have only one clustered index.

* When a table has a clustered index then the table is called a clustered table.
* If a table has no clustered index its data rows are stored in an unordered structure.’
* A table can have only one clustered index in it which will be created when the primary key constraint used in a table.

**Example for clustered index is Dictonary**

If we create any table with a primary key it will automatically consider that column as a clustered index,if we add the data in a non sequential manner it will automatically does in order wise.

Create table t1([id] int primarykey, [name] varchar(10), age int)

Insert into t1 values (3,'pk',25),(1,'rk',21),(5,'jk',27),(2,'kk',28),(4,'ok',22)

executesp\_helpindext1 – for particular table

select \* from sys.indexes---whole server.

Before creating cluster we have to delete the created cluster

Table\_name->indexes->delete the existing key

Create clustred index cluster\_name on table\_name (column conditon)

Create clustered index ix\_t1\_age on t1(age desc)

**NON CLUSTERD INDEX**

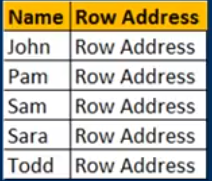
if we create a non clustered index, it will create a copy of separate index, that index contains column name which mentioned and row address of row of table which is present in database.

we can have multiple non clustered index.

For INSERT and UPDATE operations, non-clustered indexes are faster since the order of the records only needs to be updated in the index and not in the actual table.

Create Non clustred index Non cluster\_name on table\_name (column conditon)

create non clustered index ix\_t1\_age on t1(age desc)



**Example for non clustered index is content in text book.**

|  |  |
| --- | --- |
| **CLUSTER INDEX** | **NON CLUSTERD INDEX** |
| Only one clustered index in one table | We can have more than one clustered index in one table 999. |
| Clustered index fetching data is faster because it will directly search in database. | It is bit slower compared to cluster index because it will search based on column of index, if we need extra data it will go to data base where the data present by the address present in non clustered index. |
| Cluster index determines the storage order in the rows in table, it doesn’t required any additional storage. | It requires additional storage because it is stored separately from the table. |
| DML is slow bcz it has to update on order wise | order of the records only needs to be updated in the index and not in the actual table. |

**UNIQUE INDEX**

It is used to enforce the uniqueness of the key in index.

Clustered index is by default unique index, both clustered and non clustered index can be unique.

It will maintain the integrity of table, and perform fast,dosent allow duplicate records.

Unique constraints creates the unique index inside it, unique index is a proprety not a separate option.

To create The non clustered Constraints

Create unique non clustered index ix\_t1\_age on t1(age desc)

executesp\_helpindext1

To Alter The Constraints

altertable t1 addconstraint ix\_t1 uniqueclustered (age) --clustered

altertable t1 addconstraint ix\_t1 unique(age) unique and non clustered

**BY default**

Primary key constraints create a unique clustered index

select\*fromsys.indexes

Unique constraints creates a unique and non clustered index

Unique constraints or unique index can not be created in existing column which has duplicates.

**Difference b/w scan and seek**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl no** |  | **Scan Operation** | **Seek Operation** |
| **1** | **Fetches** | All the rows from the table | Selective rows from the table |
| **2** | **Touches** | Every single row of the table is either required or not | Only the required or matching row |
| **3** | **CPU Consumption** | More | Less |
| **4** | **I/O Component Usage** | More | Less |
| **5** | **Executes with** | SELECT statement | WHERE clause |

## Table Scan

* While performing table scan, the query engine starts from the beginning of the table and it goes through every row into the table. If a row matches with the criteria then it includes that into the result set.
* It is the fastest way to retrieve the data especially when there is quite a small table.
* For a small table, a query engine can load all the data in a one-shot but from a large table it is not possible i.e. more IO and more time will be required to process those large data.
* Generally, a full table scan is used when a query doesn’t have a WHERE clause.

**Index Scan**

Index scan means retriving the data by using the index.

**Index Scan is costly when**

* Table size is less
* When query requires all datas in table rows
* Extra sorts required random sorting is use.

## Index Seek

* When the search criterion matches the index well enough which can navigate directly to particular points into the data, this is known as the Index seek.
* The index seeks the fastest way to retrieve the data in the database.is used to search an index for specific values.

|  |  |  |
| --- | --- | --- |
| **SL no** | **Index Scan** | **Index Seek** |
| **1** | Used when we need to retrieve all the data such as 90% to 100% | Used when we need to retrieve some data based on some condition such as 10% of data |
| **2** | The query doesn’t have a WHERE clause and the Table doesn't have clustered index then a full Table Scan is used. | The query have a WHERE clause and the Table have clustered index then Index Scan is used. |
| **3** | The table is slower than the Index | The index is faster than Table |
| **4** | The scan is slower than Seek | Seek is faster than Scan Seek is faster than Scan |

##### **When SQL Server uses Indexes?**

##### **When ever we use where condition more to that column we use indexes.**

--before creating the clustered index for existing column, we have to delete that.

create index ix\_t1 on t1 (id asc ) -- it will create non cluster index and non unique.

create clustered index ix\_t1 on t1 (id asc ) --it will create cluster index and non unique.

create unique clustered index ix\_t1 on t1 (id asc ) --- it will create cluster index and unique.

--to create unique index, data inside column should not be repeated.

create unique index ix\_t1 on t1 (id asc ) --it will create non cluster index and unique.

alter table t1 add constraint ix\_t1 unique clustered(age)

If I’m pulling 90% of record Table Scan is better  
if I’m fetching 20% of record Index scan is better.

A Table doesn’t have any Clustered Index that table is called Heap table, it can have non cluster table.

Always read execution plans from left to right and top to bottom.

**VIEW**

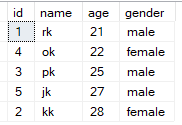
* View is a virtual table, it is a stores sql query, View doesn’t store data.
* If we execute the view it will get the data from database only.
* View is used for security, we can control the visibility of a table while creating the view we can select required columns to be shown.
* If we made any changes in views it will updated to base table.

Why we need views

If we want to repeate that sql statements multiple no of times, we can use.

Views can hide the complex data, and showing necessary data for security purpose.

select\*from t1



T1

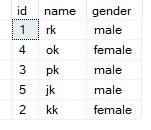
T1 is the original table,

Create view tview as

select id,name,gender from t1

view is created

select\*from tview



To View the content in view

sp\_helptext tview

TO know the space consumtion

sp\_spaceused vw\_t2;

**View can be updated under some condition**

* The select clause should not contain order by clause.
* When select clause having aggregate (summary function)function then also we cannot update, set operators .
* We can not use \* for creating view with schema binding, other wise it will work.
* We can’t create view for temp table
* We can do DML operations on view when it is create by select query only, if view created by the aggregate function we can’t use DML operations.

**Advantages of view**

* It will reduce the complexity of the database
* It is used to row and column level security

##### **What are the limitations of a View**

Views cannot be based on temporary tables.

|  |  |  |
| --- | --- | --- |
| **Sl no** | **Table** | **View** |
| 1 | Table is a physical object | View is virtual object |
| 2 | Table is independent object | View is dependent on table |
| 3 | if we create a table by using existing table, if any changes made in old table dosen't affects the new table | if we create a view by using existing table, if any changes made in table affects the view and vice versa. |

We can create the view in two ways those are

1. Simple view and Updatable views
2. Complex view and non-updatable views.

**Simple view and Updatable views**

* The view which is created based on the columns of a single table is known as the simple view.
* We can perform all DML operations on a simple view so that a simple view is also called an updatable view or dynamic view.

**complex View and non Updatable views**

* When we create a view on more than 1 table then it is known as the complex view.
* On a complex view, we cannot perform DML operations so that a complex view is also called a non-updatable or static view.

##### **Can we drop a table that has dependent views on it? = yes**

##### **Can we create a view based on other views? = yes**

##### **What are indexed views? Or what are materialized views?**

When we create an index on a view, the data gets physically stored in the view. So, when we issue a select query against an indexed view, the data is retrieved from the index without having to go to the underlying table, which will make the select statement to work slightly faster. However, the disadvantage is DML operations will become a little slow, because every time we insert or delete a row from the underlying table, the view index needs to be updated. In short, DML operations will have a negative impact on performance.

create view vw1

with schemabinding

as

select id,name,gender from dbo.t2

select \* from vw1

create unique clustered index ind on vw1(id)

* for creating index view we need to create the views with schema binding.
* For creating schema bindning schema name is mandatory like dbo is default.
* Only unique clustered index can be created.

**T-SQL**

SQL a non procedural programming language .

**TRANSACTION**

A group of sql commands that stored as single unit,it changes the data stored in a database.

A successful transaction must pass ACID test.

**Transaction should follow the ACID property**

**Atomicity-**all the statements in the transactions either completed or they rolled back. It should not be half done.

**Consistency:**  It means that whatever happens in the middle of the transaction, this will never leave your database in a half-completed state.

* If the transaction is completed successfully, then it will apply all the changes to the database.
* If there is an error in a transaction, then all the changes that have already been made will be rollback.

**Isolation**: Isolation ensures that the execution of transactions concurrently does not interfere with each other. Each transaction should operate independently of and be unaware of other transactions executing concurrently.

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

**Durable** – Once the transaction is completed, then the changes it has made to the database will be permanent, system failures or restarts do not affect committed transactions

**COMMIT**

begintransaction

deletefrom [Transaction\_Tb2] where TranAmt=7000

COMMIT

This query is in processing format, neither the deleted data is present nor deleted, if we give commit it will save the executed part permanently.

**ROLLBACK**

select\*from [dbo].[Transaction\_Tb2]

begintransaction

deletefrom [Transaction\_Tb2] where TranAmt=7000

deletefrom [Transaction\_Tb2] where TranAmt=15000

rollback

This query is in processing format, neither it is present nor deleted, if we roll back it will undo the changes.

If we use rollback command it will roll back everything.

**SAVEPOINT**

select\*from [dbo].[Transaction\_Tb2]

begin transaction

save transaction s1

delete from [Transaction\_Tb2] where TranAmt=7000

save transaction s2

delete from [Transaction\_Tb2] where TranAmt=15000

rollback transaction s2

This query is in processing format, neither it is present nor deleted, if we give commit it will save the executed part, if we roll back it will undo the changes.

If we use rollback with save point it will rollback till mentioned save point, not everything command it will roll back everything.

begin

begin try

begin transaction

update Transaction\_Tb2 set tranamt=11000 where custid=1001 and TraniD=20001

print'done1'

update Transaction\_Tb2 set trandate='2020-02-30'where custid=1001 and TraniD=20002

print'done 2'

commit transaction

endtry

begin atch

rollback transaction

print'rollback done'

end catch

end

here if first update is done, and second update can’not be done due to errors, it will get rollback every thing. This is example for unit

**Variable**

**Variable** is name given to particular memory location to store some data’s

**Syntax for declaring the variable in T-SQL**

Declare @variable\_name

Set @variable name= values

Offset, fetch next both of this order by is mandatory.

In fetch we can mention the from which rows data should be fetch

**IF CONDITION**

If given condition is satisfy it will execute one set of code, other wise another set of code.

**If condition**

**Begin**

**sql statement**

**end**

**else**

**Begin**

**sql statement**

**end**

if(@abc isnull)

print'@abc is null'

else

print'@abc is not null'

**While Loop**

Repeat the task for multiple number of times unitll the given fail

**while (condition)**

**begin**

**sql sataemnts**

**incremental condition**

**end**

declare @id int

set @id=65

while (@id<=90)

begin

print char(@id)

set @id=@id+1

end

It will print A - Z

**Stored procedure**

* A stored procedure is a precompiled SQL query with an assigned name and directly saved in the database.
* Server can accept input parameter.
* It will use system storage.
* Procedure means set of instructons.

**Why we need**

Whenever we want to execute a SQL query from an application the SQL query (statement) what we send from an application will be first compiled(parsed) for execution, where the process of compiling(parsing) is time-consuming because compilation occurs each time when we execute the query or statements.

**The Parameters of a stored procedure can be of two type**

1. Input parameters
2. Output parameters

The **Input parameters** are basically used for bringing value into the procedure for execution whereas the **output parameters** are used for carrying a value out of the procedure after execution.

**Syntax**

Create procedure sp\_procedure\_name

As

Begin

Sql queries

End

**To view stored procedure name**

Sp\_helptext stored procedure name

**Advantage of SP**

* Code Reusable will be increased.
* Easier to modify
* Security purpose by using with encrypted.
* Increase the performance.

**Disadvantage**

Difference b/w view and SP

System memory v/s database memory

Return v/soutput

**Getting data through Stored procedure by input paramater**

Creation of stored procedure

Create procedure sp\_sha

@id int

as

begin

select\*from sha

where id=@id

end

retrieving of data through stored procedure

**sp\_sha 1** (it will retrieve the id =1 datas)

**sp\_sha 2** (it will retrieve the id =2 datas)

**sp\_sha 3** (it will retrieve the id =3 datas)

update the values through Stored procedure

create procedure sp\_sha\_update

@id int,

@sal int

as

begin

update sha set sal=@sal

where id=@id

end

sp\_sha\_update 4,2100 (it will update the values)

**Syntax fo alter SP**

Alterprocedure sp\_procedure\_name

As

Begin

Sql queries

End

alterprocedure sp\_outsha

@id int,

@output int output

as

begin

select @output =count(sname)from sha

where id=@id

end

**TO encrypt the SP**

createprocedure sp\_sha\_update

@id int,

@name varchar(20)

withencryption

as

begin

update sha set sname=@name

where id=@id

end

sp\_helptext sp\_sha\_update

**Getting Output By Stored Procedure**

createprocedure sp\_sha

@id int,

@outintoutput

as

begin

select @gets =count(sname)from sha

where id=@id

end

declare @abc int

execute sp\_sha2, @abc output

if(@abc isnull)

print'@abc is null'

else

print'@abc is not null'

print @abc

create or alter proc sp\_outpt

@reg varchar(20),

@r varchar(20) output,

@sales int output

as

begin

select @r= Region,@sales=sum(Sales) from Orders

where region = @reg

group by Region

end

declare @r1 varchar(20), @sl int

exec sp\_outpt central,@r1 out,@sl out

print @r1 + ' is ' + cast( @sl as varchar)

**Single input parameter should accept multiple values in stored procedure**

create or alter proc sp\_reg\_multi\_dynmc

@re varchar(20),

@sg varchar(20)

as

begin

select \* from Orders

where Region in (select \* from string\_split( @re,',')) and segment in (select \* from string\_split( @sg,','))

end

exec sp\_reg\_multi\_dynmc 'west,central','corporate,consumer'

|  |  |  |
| --- | --- | --- |
| **Sl no** | **Sp** | **View** |
| 1 | it is precompiled | it is not precompiled |
| 2 | it will accept input parameter | it won't accept input parameter |
| 3 | we have to execute it without select clause | we have to execute it with select clause |

4 it stores the result it fetches from table

**Cursor**

It will retrieve the data row by row from a result set one row at a time,A database cursor will traversal over the rows of a result set.

It is used to perform a DML operation on records row by row,

It is temporary memory, it will execute individually row by row.

**Two Types Of Cursors**

**Implicit Cursor :**it is also called as default cursor in ms sql, these cursors allocate by sever when user perform DML operations. An implicit cursor is also created by the system when the SELECT query selects the single row.

**Explicit Cursor :**An explicit cursor holds multiple records but processes a single row at a time. It uses the pointer, which moves to another row after reading one row.

**Methods of cursor**

**Next :** it will display the next record.

**Prior :** it is used to fetch the previous record.

**First :** Returns the first row in the cursor and makes it the current row.

**Last :** Returns the last row in the cursor and makes it the current row.

**Absolute n :**This method allows the system to access the data of the exact nth row from the cursor table.

**RELATIVE { n or @nvar}**

This method allows the system to access the data in both incremental and decremental processe.  
If n or @nvar is positive, returns the row n rows beyond the current row and makes the returned row the new current row. If n or @nvar is negative, returns the row n rows prior to the current row and makes the returned row the new current row. If n or @nvar is 0, returns the current row.

-- Execute the SELECT statement alone to show the

-- full result set that is used by the cursor.

SELECT LastName, FirstName FROM Person.Person

ORDERBY LastName, FirstName;

-- Declare the cursor.

DECLARE contact\_cursor SCROLLCURSORFOR

SELECT LastName, FirstName FROM Person.Person

ORDERBY LastName, FirstName;

OPEN contact\_cursor;

-- Fetch the last row in the cursor.

FETCH LAST FROM contact\_cursor;

-- Fetch the row immediately prior to the current row in the cursor.

FETCH PRIOR FROM contact\_cursor;

-- Fetch the second row in the cursor.

FETCH ABSOLUTE 2 FROM contact\_cursor;

-- Fetch the row that is three rows after the current row.

FETCH RELATIVE 3 FROM contact\_cursor;

-- Fetch the row that is two rows prior to the current row.

FETCH RELATIVE -2 FROM contact\_cursor;

CLOSE contact\_cursor;

DEALLOCATE contact\_cursor;

**Offset :** It will skip till 10 rows and retrieve the data from 11th row, order by is mandatory.

select\*from dbo.employees

orderby employee\_id

offset 10 rows

**Fetch Next**: it will fetch next rows only which is mentioned, for fetch next, offset is mandatory.

select\*from dbo.employees

orderby employee\_id

offset 10 rows

fetchnext 10 rows only

**Fetch first**: it will fetch first rows only which is mentioned, for fetch next offset is mandatory.

select\*from dbo.employees

orderby employee\_id

offset 10 rows

fetchfirst 10 rows only

**Fetch Status**

This function returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection.

while@@FETCH\_STATUS=0

|  |  |
| --- | --- |
| 0 | The FETCH statement was successful. |
| -1 | The FETCH statement failed or the row was beyond the result set. |
| -2 | The row fetched is missing. |
| -9 | The cursor is not performing a fetch operation. |

**SQL cursor life cycle**

**Declaring cursor**- cursor is declared by defining the sql statements.

**Opening cursor** – to opens the cursor for storing the data retrieved from the result set.

**Fetching cursor**- rows can be fetched from the cursor row by row

**Closing cursor**- cursor should be close explicitly after data manipulation

**Deallocating cursor**- to delete the cursor definition and release all the system memory.

declare the variable which we want to retrieve from the table

declarecursor

sysntax

declare cursor\_name cursorforsql queries

opencursor

syntax

open cursor\_name

fetchdata

fetchnextfrom cursor\_name into @declared\_variable

@while condition

begin

print

fetchnextfrom cursor\_name into @declared\_variable

close cursor\_name

deallocate cursor\_name

declare @eid int,@ename varchar(20),@job varchar(20),@sal int

declare cur\_name cursorforselect employee\_id,first\_name,job\_id,sum(salary)from employees

groupby employee\_id,first\_name,job\_id

open cur\_name

fetchnextfrom cur\_name into @eid ,@ename ,@job,@sal

WHILE@@FETCH\_STATUS<>-1

begin

printcast(@eid asvarchar)+' '+@ename +' '+cast(@job asvarchar)+' '+cast(@sal asvarchar)

fetchnextfrom cur\_name into @eid ,@ename ,@job,@sal

end

close cur\_name

deallocate cur\_name

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

select\*from sha

declare @sid int,@ssal int, @sname varchar(20)

declare c2 cursorforselect id,sal,sname from sha

open c2

fetchnextfrom c2 into @sid,@ssal,@sname

while@@FETCH\_STATUS=0

begin

update sha set sname='pradeep'where id=4

fetchnextfrom c2 into @sid,@ssal,@sname

end

close c2

deallocate c2

## Basic Type of Cursor

1. STATIC Cursor
2. Forward Only cursor
3. KEYSET Driven Cursor
4. Dynamic Curs

### STATIC Cursor

* The static cursor can move in forward as well as backward direction. Compared to other cursors, it is slow and uses more space in the memory.
* By default, these types of cursors are scrollable. The static cursor does not allow the database users to modify and delete data.

### Forward Only Cursor

* This type of cursor accesses and updates the data only in the forward direction through the result set. So, it is the fastest cursor among all the four cursors.
* The main disadvantage of this cursor is that it does not support backward scrolling.

Following are the three types of 'Forward only Cursor':

1. Forward\_Only KEYSET,
2. FAST\_FORWARD

### KEYSET Driven Cursor

This type of cursor accesses the data from the first row to the last row and last to the first row. When the user opens the KEYSET cursor, it automatically creates the list of unique values which uniquely identify each row in the entire result set.

### DYNAMIC Cursor

* The dynamic cursor is just opposite to the static cursor. It allows us to execute the INSERT, DELETE, and UPDATE operations while the cursor is open.
* It checks all the modifications done on the rows and values in the result set.

**Scroll Cursor:**

A Scroll Cursor is a SQL Object, or we can say like a Virtual table that retrieves data from the table one row at a time by scrolling first, next, last, absolute and prior position .  We can fetch any record as first, last, prior and specific record from the table.

declare scroll\_cursor cursor  
scroll for  
select \* from emp.

## How can we avoid cursors?

* **Using the SQL while loop**
* **User-defined functions**
* **Using Joins**

**Functions in SQL Server**

Function is a set of sql statements that performs a specific task.

If we want to repeatedly write large sql scripts to perform the same task , we can create a function that performs that task.

Next time instead of writing query, we can simply call function.

Function is a stored program that we can pass parameters and return values.

We can create the stored procedure and execute the task multiple no of times, but we can’t use stored procedure inside the sql statements.

**Rules to create the functions.**

* Function name should not be start with special characters.
* Functions only works with select statements.
* Functions can be used anywhere in sql
* Functions compile every time.
* Functions only work with input parameters.
* Try and catch are not used in functions.

**SQl Function Types**

1. User Defined - create by users.

* Scalar function
* Inline table valued
* Multi-statement table valued functions.

1. System Defined – built in functions.

**Scalar function**

* Sql server scalar function takes zero or more parameters and returns single (scalar)value.
* The returned value can be of any data type, except text,cursor,image,timestamp.
* If else blocks and while loops can be used.
* It can call other functions.
* It can’t update data

Steps

Create the function without parameter.

createfunction Showmessage()

returnsvarchar(100)

as

begin

return'Jai shree Raam'

end

select dbo.Showmessage() -----for the execution dbo is mandatory, data base owner.

dropfunction show\_text;

--how many emp belong to a dept

createoralterfunction fn\_findDEpt(@dept\_id int)

returns int

as

begin

return

(selectcount(\*)from employees

where department\_id=@dept\_id)

end

select dbo.fn\_findDEpt(50)

create function checkVoterAge(@age asint)

returns varchar(100)

as

begin

declare @str varchar(100)

if @age >= 18

begin

set @str='right to vote'

end

else

begin

set @str='not ready to vote'

end

return @str

end

select dbo.checkVoterAge(18)

**Inline table valued function**

* It returns a table set.
* We have to specify Table as the return type, instead of any scalar data type like int, varchar etc.
* There is no begin and end blocks.
* Tables that gets returned, is determined by the select command within the function.

createfunction fn\_id(@id int)

returnstable

as

return(select\*from tblemployee where id=@id)

select\*from fn\_id(1)

for update

create function pk2()

returns table

return (select\*from tblemployee )

update pk2()set name='tim'where id=1

select\*from pk2()

**Multi-statement table valued functions**

Create function pk1()

returns @tk table (id int,namevarchar(20))

as

begin

insert into @tk

select id,name from tblemployee

return

end

select\*from pk1()

|  |  |
| --- | --- |
| **Inline** | **Multi** |
| The returns clause cannot contain the structure of the table. | We can specify the structure of the table that gets returned. |
| It don’t have begin and end block | It has begin and end block |
| Performance is better, bcz sql server treats the inline as a view. | Performance is not so better as inline, bcz sql server treats the inline as a Stored procedure. |
| We can update the underlying (base)table | We can’t update the table |

Deterministic function – it will returns same result any time they called with a specific input.

Ex. Square(),power(),sum(),avg(),count().

Non Deterministic function – it may returns different results each timethey called with a specific set of input values.

Ex – getdate().

Rand() is a non-deterministic function, if we provide a value it become Deterministic function.

selectrand(395)

**Encryption in function**

createoralterfunction pk2()

returnstable

withencryption

return (select\*from tblemployee )

sp\_helptext pk2

**With SCHEMABINDING**

SCHEMABINDING keywords prevent tables used in the views or user defined functions to make any such modifications that can affect the view’s and defined functions definition.

createoralterfunction pk2()

returnstable

withSCHEMABINDING

return (select\*fromdbo.tblemployee )

sp\_helptext pk2

|  |  |  |
| --- | --- | --- |
| **Sl no** | **Sp** | **Function** |
| 1 | it is precompiled once | it should precompiled every time |
| 2 | it has input and output parameter | it has input parameter |
| 3 | we have to execute it with execute command | we have to execute it with select clause |
| 4 | Functions can be called from Procedure | Procedures cannot be called from a Function. |
| 5 | we can use DML in SP | will not support DML operation |
| 6 | SP may or may not return value | Function returns value |

**TRIGGERS**

Trigger is a set procedure that automatically executes when an event occurs in the database server.

Triggers are used to protect the data integrity(maintaining the quality of the data) in the database.

**Types of trigger**

* DML trigger
* DDL trigger
* LOGON Trigger

DML Triggers are fired automatically in response to DML events(insert,update,delete)

**DML Types**

* **After triggers(for triggers)**
* After insert - This event is called when the new row is entered in the table.
* After update - This event is called when the existing record is changed or modified in the table.
* After delete- This event is called when the existing record is changed or modified in the table.
* **Instead of triggers**

## Advantages of Triggers in SQL

1. It maintaining the data and referential integrity in the tables.
2. Triggers helps in executing the scheduled tasks because they are called automatically.
3. It will track the every moment happens in table.

## Disadvantages of Triggers in SQL

1. They are not compiled.
2. It is not possible to find and debug the errors in triggers.
3. If we use the complex code in the trigger, it makes the application run slower.
4. Trigger increases the high load on the database system.

**CREATE TRIGGER**

--============================after insert================================

Create or alter trigger tr\_student\_for\_insert

on sha

after insert

as

begin

print'data inserted'-- it will print message

select\*from inserted-- it is magical table, it will returns row with complete data

end

--=========================================================================

Create table auditing(id int identity(1,1),audt varchar(50))

Drop table auditing

Truncate table auditing

Select \* from auditing

create or alter trigger tr\_audit on sha

after insert

as

begin

declare @id int

select @id = id from inserted

insert into auditing values ('id inserted '+cast(@id asvarchar)+'data inserted in '+cast(getdate()asvarchar))

select \* from auditing

end

--==============================================================================

Create trigger tr\_deleteData\_auditing on sha

after delete

as

begin

declare @id int

select @id=id from deleted --- deleted is also a magic table

insert into auditing values ('id inserted '+cast(@id asvarchar)+' data inserted in '+cast(getdate()asvarchar))

select \* from auditing

end

=============================== after update =============================

create trigger tr\_update\_sha on sha

after update

as

begin

select \* from inserted

select \* from deleted

end

**Instead of Triggers**

This trigger will fire instead of the respective statement execution of DML operations.

Create trigger tr\_insteadOf\_insert on sha

instead of insert

as

begin

print'can not update '

end

droptrigger tr\_insteadOf\_insert

---------------------------delete----------=========

Create trigger tr\_insteadDElete on sha

instead of delete

as

begin

print'u are not allowed to delete'

end

**Exception handling**

An error condition during program execution is called an exception.

* We have to write the code which gives the exception inside the try block and the message for that has to be written in catch block.
* If exception occurs catch will execute, other wise won’t execute.
* After exception occurs it wont execute next line, directly goes to catch.

begin

begin try

select 10/0

end try

begin catch

print 'fool'

select @@ERROR as [@@ERROR],

ERROR\_NUMBER() as [ERROR\_NUMBER],

ERROR\_STATE() as [ERROR\_STATE],

ERROR\_LINE() as [ERROR\_LINE],

ERROR\_MESSAGE() as [ERROR\_MESSAGe],

ERROR\_PROCEDURE() as [ERROR\_PROCEDURE],

ERROR\_SEVERITY() as [ERROR\_SEVERITY]

end catch

end

C:\Users\Admin\Desktop\err.PNG

**Temporary tables**

A temporary table is copy of base table stored in tempdb exists only while the database session.

* Temporary tables provide temporary data storage in extract form of original tables for quick access of data.
* Temporary tables are stored in TempDB.
* hey work like a regular table in that you can perform the operations select, insert and delete as for a regular table.
* If created inside a stored procedure, they are destroyed upon completion of the stored procedure.
* if we done any operation on temp table it won't affect on base table

|  |  |  |
| --- | --- | --- |
| **SL NO** | **Temp table** | **Base table** |
| 1 | It is created in TempDb | It is created in Database |
| 2 | temp tables are faster in loading data | Base tables are slower in loading data |
| 3 | it will delete automatically when session is end or server closed | it will store permanently untill delete it by manually and |
| 4 | IT is created by using prefix as # | IT don't use any prefix as # to create table |

**Types of temporary table**

1. Local temporary table
2. Global temporary table

### Local Temporary Tables

Local temporary tables are the tables stored in tempdb. Local temporary tables are temporary tables that are available only to the session that created them. These tables are automatically destroyed at the termination of the procedure or session. They are specified with the prefix #.

**Syntax**

Create table #table\_name ( column\_name varchar(20), column\_no int )

### Global Temporary Tables

Global temporary tables are also stored in tempdb.Global temporary tables are temporary tables that are available to all sessions and all users. They have dropped automatically when the last session using the temporary table has been completed. They are specified with the prefix #, for example, ##table\_name.

Create table **#**table\_name

createtable demo11( id intnotnullunique,namenvarchar(20)notnull, age intnotnull, DOB datenotnull)

from one table to another table

select\*into #tem from sha

select id,name into #temptable from emp.

select \* from #table\_name– to retrieve the details of table

we use # for in the temporary table.

* It is stored inside->database->system database->tempDB->temporary tables.
* Local temporary table is available for connection which is created, if we use another connection it wont work even database base is same.
* Connection means new query sheet.
* Local temporary table is automatically delete when connection is closed, we can also drop table.
* We can use same name for temporary table, sql internally uses a random address number to differentiate the table names.
* We can perform operations like ddl, dml and transactions also. we can’t use stored procedures

It is very beneficial to store data in SQL Server temp tables rather than manipulate or work with permanent tables. Let’s say you want full DDL or DML access to a table, but don’t have it. You can use your existing read access to pull the data into a SQL Server temporary table and make adjustments from there. Or you don’t have permissions to create a table in the existing database, you can create a SQL Server temp table that you can manipulate. Finally, you might be in a situation where you need the data to be visible only in the current session.

|  |  |
| --- | --- |
| **Local temporary table** | **Global temporary table** |
| It is prefixed with single # | It is prefixed with double## |
| Local is limited to only server which is created | Global is visible to all server |
| Local is automatically droped when local server closed | Global will delete at the last server is closed. |
| We can create a multiple local temporary table with same names | We can create only one table |

**Temporary tables are allowed CREATE INDEXes whereas, Table variables aren't allowed CREATE INDEX**

Table variable

@tale\_name

* Create table @table\_name(col name, data type, constrinats)
* Dml operations can be performed.
* Transactions can not used, we canuse stored procedures
* The TRUNCATE statement helps to delete all rows in the tables very quickly. However, this statement cannot be used for table variables.

**Temp Variables:**Temp Variables are also used for holding the data fora  temporary time just like Temp tables. Temp variables are created using “DECLARE” statements and are assigned values by using either a SET or SELECT command. After declaration, all variables are initialized as NULL, unless a value is provided as part of the declaration. This acts like a variable and exists for a particular batch of query executions. It gets dropped once it comes out of batch. Temp variables are also created in the Tempdb database but not the memory.

1. **Declare** **Temp** variable:
2. **Declare** @My\_var2TABLE
3. (
4. IIDint,
5. NameNvarchar(50),
6. SalaryInt,
7. City\_NameNvarchar(50)
8. )

Insert result set of another query into temp variable,

1. Insert Into @My\_var2
2. **Select**\* from Employee
3. **WHERE** IID<8 AND Salary>20000

**Retrieve data from temp variable:**

1. **Select**\***from**@My\_var2

Write a query with schema binding that contain all the records od table

**What is SQL logging?**

What Is a SQL Server Log File? A SQL Server log file is a transaction log file that records all database transactions and modifications. In SQL terms, this log file records all the INSERT, UPDATE, and DELETE query operations performed on a database.

Subset of SQL – DDL,DML,DQL,TCL,DCL.

Case manipulation – UPPER, LOWER, INITCAP

Show the total amount of male patients and the total amount of female patients in the patients table.  
Display the two results in the same row.

SELECT

(SELECT count(\*) FROM patients WHERE gender='M') AS male\_count,

(SELECT count(\*) FROM patients WHERE gender='F') AS female\_count;

select

sum(case when gender = 'M' then 1 end) as male\_count,

sum(case when gender = 'F' then 1 end) as female\_count

from patients;

WITH CTE AS

(select GENDER, COUNT(\*) AS Male\_count FROM patients

WHERE gender = 'M'

group by gender),

cte2 as

(select GENDER, COUNT(\*) AS FeMale\_count FROM patients

WHERE gender = 'F'

group by gender)

SELECT a.Male\_count, b.FeMale\_count FROM CTE a cross join CTE2 b

Show all of the patients grouped into weight groups.  
Show the total amount of patients in each weight group.  
Order the list by the weight group decending.  
  
For example, if they weight 100 to 109 they are placed in the 100 weight group, 110-119 = 110 weight group, etc.

select count(patient\_id) as pt\_in\_grp,weight\_grp from (select \*,

case

when weight between 0 and 9 then 0

when weight between 10 and 19 then 10

when weight between 20 and 29 then 20

when weight between 30 and 39 then 30

when weight between 40 and 49 then 40

when weight between 50 and 59 then 50

when weight between 60 and 69 then 60

when weight between 70 and 79 then 70

when weight between 80 and 89 then 80

when weight between 90 and 99 then 90

when weight between 100 and 109 then 100

when weight between 110 and 119 then 110

when weight between 120 and 129 then 120

when weight between 130 and 139 then 130

when weight between 140 and 149 then 140

when weight between 150 and 159 then 150 end as weight\_grp from patients)a

group by weight\_grp

order by weight\_grp desc

find emp where sal containd his deptno

SELECT \* FROM emp

WHere CHARINDEX(cast( substring(cast(deptno as varchar),1,1) as varchar),cast(sal as varchar))>0

SELECT \*FROM emp

WHERE sal LIKE '%' + substring(cast(deptno as varchar),1,1)+ '%';

create table emple

(

emp\_id int,

dept\_id int

)

insert into emple values (1,8),(2,8),(3,8),(4,7),(5,9),(6,9)

select \* from emple;

with cte as

(select dept\_id,count(dept\_id) cnt from emple

group by dept\_id)

select b.emp\_id,a.cnt from cte a join emple b on a.dept\_id=b.dept\_id

order by b.emp\_id

create table tree

(

id int,

p\_id int

)

insert into tree values(1,null),(2,1),(3,1),(4,2),(5,2),(6,3)

drop table tree

select \* from tree;

select id, case

when p\_id is null then 'Root'

when id in (select p\_id from tree where p\_id is not null ) then 'Inner'

else 'Leaves' end as Steps from tree

hhiiiiiiiiiiiiiiiiiiiiiiiii