1)creating database

create database [practice]

2)use existing database

use [practice]

3)create table

create table create\_table(

id int,

name varchar(100),

gender varchar(100),

salary int,

age int

)

4)creating random data to practice

create table employee

(

  id int,

  emp\_name varchar(100),

  gender varchar(100),

  salary int,

  dept\_id int,

  location\_id int

 )

declare @count as int=1

while(@count<=250)

begin

    declare @id as int

    declare @emp\_name as varchar(100)

    declare @gender as varchar(100)

    declare @dept\_id as int

    declare @salary as int

    declare @location\_id as int

    set @id=@count

    set @emp\_name='employee '+cast(@count as varchar(10))

    set @gender=iif(round(rand()\*1,0)+1=1,'female','male')

    set @dept\_id=round(rand()\*10,0)+1

    set @location\_id=round(rand()\*5,0)+1

    set @salary=round(rand()\*100000,-3)

    insert into employee values(@id,@emp\_name,@gender,@salary,@dept\_id,@location\_id)

    set @count=@count+1

   end

select \* from employee

update employee set dept\_id=20 where dept\_id=11

update employee set dept\_id=20 where dept\_id=6

go

create table department

(

dept\_id int,

dept\_name varchar(100),

dept\_head\_id int

)

declare @count as int=1

while(@count<=13)

begin

    declare @dept\_id as int

    declare @dept\_name as varchar(100)

    declare @dept\_head\_id as int

    set @dept\_id=@count

    set @dept\_name='dept '+cast(@count as varchar(10))

    set @dept\_head\_id=@count\*(round(rand()\*20,0)+1)

    insert into department values(@dept\_id,@dept\_name,@dept\_head\_id)

    set @count=@count+1

end

select \* from department

go

create table location

(

location\_id int,

location\_name varchar(100),

location\_head\_id int

)

declare @count as int=1

while(@count<=8)

begin

    declare @location\_id as int=@count

    declare @location\_name as varchar(100)='location '+cast(@count as varchar(10))

    declare @location\_head\_id as int =@count\*(round(rand()\*25,0)+1)

    insert into location values(@location\_id,@location\_name,@location\_head\_id)

    set @count=@count+1

end

select \* from location

go

create table join\_date

(

id int,

join\_date date

)

declare @count as int=1

while(@count<=250)

begin

    declare @id as int=@count

    declare @date as date

    set @date=dateadd(day,round(rand()\*5000,0),cast('2012-02-03' as date))

    insert into join\_date values(@id,@date)

    set @count=@count+1

end

select \* from join\_date

5)primary key and foreign key

primary key is a column or group of columns which are used to uniquely identify the records in the table.

primary key should be not null and unique.

foreign key is a column or group of columns which is a primary key in the other table.

by using primary key and foreign key we maintain the relation between the table.

6)where clause used to filter the data based on a condition before data aggregated. use it before group by clause

select \* from employee where salary>50000 --salary greater than 50000

select \* from employee where salary<50000 --salary less than 50000

select \* from employee where salary>=50000 --salary greater than equal to 50000

select \* from employee where salary<=50000 --salary less than equal 50000

select \* from employee where salary=5000 --salary equal to 50000

select \* from employee where salary between 20000 and 60000 --salary between 20000 and 60000

select \* from employee where salary<>50000 --salary not equal to 50000

select \* from employee where salary in (20000,30000,50000,60000) --salary in given multiple values

select \* from employee where salary not in (20000,30000,50000,60000) --salary not in given multiple values

select \* from employee where salary>50000 and location\_id>=5 --validates multiple conditions if all conditions satisfies then only we get output.

select \* from employee where salary>50000 or location\_id>=5 --validates multiple conditins if any one conditions satisfies then we get output.

7)order by clause

--order by clause used to order the data in the table it always at last clause. we can use alias name in order by clause.

select \* from employee order by salary  --it order table by salary in asc by default asc we no need to mention

select \* from employee order by salary desc --it order table by salary in descending by default asc we no need to mention

select \* from employee order by salary asc,dept\_id desc --it order table by salary in asc and dept\_id in desc

select \* from employee order by 4 --we can refer columns by it position 4 means 4th column i.e. salary asc.

select \* from employee order by 4 desc --we can refer columns by it position 4 means 4th column i.e. salary desc.

select \* from employee order by 4 asc, 5 desc --it order table by 4 column asc i.e. salary 5 column desc i.e dept\_id.

--we cannot use order by in view,sub query, cte untill we use offset/fech next.

8)like operator

/\*like use for regex matching.

\_ for single character

% for any number of characters

[a,b,c] for mutiple characters

[a-z0-9] range of values

To use % and \_ in like use them in []

^ not operator\*/

select \* from employee where emp\_name like 'e%' --start with e

select \* from employee where emp\_name like '%5' --end with 5

select \* from employee where emp\_name like '\_m%' --second character m

select \* from employee where emp\_name like '[e,r,t]%' --start with e,r or t.

select \* from employee where emp\_name like '[a-z]%[0-9]' --start with alphabet and end with digit

select \* from employee where emp\_name like '[^a-z]%[0-9]' --not start with alphabet and end with digit

select \* from employee where emp\_name not like '%5' --not end with 5

--^ is not for each pattren where not used for total pattren.

declare @name as varchar(100)='pavan%kumar\_'

Select @name where @name like '%[%][\_]'

9)distinct used to get distinct records

select distinct(dept\_id) from employee --to get distinct values from single column

select distinct dept\_id,location\_id from employee --to get distinct values from multiple columns

10)constraint

/\*not null it does not allows nulls.

null it allows nulls

check it allows value with in specified limit

unique it does not allows duplicates

default it assign default value if value not given

primary key used to uniquely identifies the records in the table.

foreign key is a primary key in other table.\*/

11)implementing constraints.

--with out constraint name

create table constraint1

(

id int primary key,

name varchar(10) unique,

gender varchar(10) check(gender in ('m','f')),

salary int default 200000,

age int not null,

dept\_id int foreign key references [location](location\_id)

)

--with constraint name

create table constraint2

(

id int constraint pk primary key,

name varchar(10) constraint uk unique,

gender varchar(10) constraint ck check(gender in ('m','f')),

salary int constraint dk default 200000,

age int constraint nn not null,

dept\_id int constraint n  null,

location\_id int constraint fk foreign key references [location](location\_id)

)

--pk and fk at last

create table constraint3

(

id int,

name varchar(10) constraint uk2 unique,

gender varchar(10) constraint ck2 check(gender in ('m','f')),

salary int constraint dk2 default 200000,

age int constraint nn2 not null,

dept\_id int,

constraint pk2 primary key(id),

constraint fk2 foreign key(dept\_id) references [location](location\_id)

)

--after creating table implementing constraint

create table constraint4

(

id int,

name varchar(10),

gender varchar(10),

salary int,

age int,

dept\_id int,

location\_id int

)

alter table constraint4 alter column id int not null

alter table constraint4 add constraint pk4 primary key(id)

alter table constraint4 add constraint uk4 unique(name)

alter table constraint4 add constraint df default 20000 for salary

alter table constraint4 add constraint fk foreign key(locatin\_id) references [location](location\_id)

alter table constraint4 alter column dept\_id int null

alter table constraint4 add constraint ck4 check(gender in ('m','f'))

12)alter command

--alter is a DDL command which used to modify the structure of table.

alter table constraint4 alter column dept\_id float --to change column datatype.

alter table constraint4 add cgpa float --add column

alter table constraint4 drop column cgpa --to drop column

alter table constraint4 drop constraint ck4 --to drop constraint

alter database rev modify name=revision --to alter databse name

sp\_rename ‘rev’ ,’revision’ --rename database

13)aggregate functions, group by, having

/\*group by used to group the data at specified column level in select statement.

having used to filter the data after aggregating

aggregate functions are use to aggregate the data there are different aggregate functions

sum(),min(),max(),count(),avg() etc.\*/

select location\_id,sum(salary) as sum\_sal,count(salary) as count\_salary,avg(salary) as avg\_salary,

min(salary) as min\_salary,max(salary) as max\_salary from employee group by location\_id having sum(salary)>200000 and count(\*)>3

14)top percent alias

/\*top used to select secified number of top records from table.

percent used to represent the percentage of records like 20 percent means 20 percentage of total records.

alias used to give alias name to column using as keyword.\*/

select top 10 \* from employee

select top 10 percent \* from employee

select dept\_id,sum(salary) as sum\_sal from employee group by dept\_id

select dept\_id,sum(salary) as 'sum\_sal' from employee group by dept\_id

--we can kept alias name directly or in single codes.

15)order of sql query written

select

from

where

group by

having

order

offset/fetch next

16)order of execution

from

where

group by

having

select

order

offset/fetch next

17)offset/fetch next

--it used to limit the number of rows to display.

--using offset fetch next we can use order by in cte, subquery and view.

select \* from employee order by salary desc

offset 10 rows  --number of rows to skip

fetch next 10 rows only --number of rows to fetch

--offset/ fetch next can used with the order by clause if order by clause is not present it return error.

18)advanced joins

select id,emp\_name,gender, salary,e.dept\_id,dept\_name,dept\_head\_id into emp\_head from employee e join deptartment d on e.dept\_id=d.dept\_id

select \* from employee e full join department d on e.dept\_id=d.dept\_id where e.dept\_id is null or d.dept\_id is null --non matching from both

select \* from employee e full join department d on e.dept\_id=d.dept\_id where e.dept\_id is null --non matching from both

select \* from employee e full join department d on e.dept\_id=d.dept\_id where e.dept\_id is null --records in b which are not in a.

select \* from employee e full join department d on e.dept\_id=d.dept\_id where d.dept\_id is null --record in a not in b.

select e1.\*,e2.emp\_name as manager from emp\_head e1 join emp\_head e2 on e1.dept\_head\_id=e2.id order by e1.dept\_id --self join

19)subquery

/\*writing a query in a query is called subquery.

there are two types of sub query.

correlated and non correlated sub query.

sub query executes independently with out main query.

sub query does not executes independently with main query.for each row in outer query inner query executes onces.\*/

select id,emp\_name,dept\_id,gender,(select dept\_head\_id from deptartment d where d.dept\_id=e.dept\_id) as dept\_head from employee e --correlated sub query(using sub query in seelct clause)

select id,emp\_name,dept\_id,gender from employee where dept\_id in (select dept\_id from employee)--non correlated sub query (using sub query in where clause)

select \* from (select dept\_id,dept\_head\_id from deptartment) d --using subquery in from clause

select dept\_id,sum(salary) as sal from employee group by dept\_id having sum(salary)>=(select avg(salary) from employee) --sub query in having clause

20)case statment

--case statement used to give output based on validating the condition we given in it.

declare @name as varchar(10)='pavan'

select case when @name='pavan' then 'kumar' else 'thota' end as [name]

select \*,

case when salary>80000 then 'high'

    when salary>50000 then 'medium' else 'low' end as salary\_range

from employee

--use aggregate functions with case.

select dept\_id,

count(case when salary>=50000 then 1 else null end) as high\_sal\_count,

sum(case when salary>=50000 then salary else 0 end) as sum\_sal\_high

from employee group by dept\_id

21)union and union all.

union retuns combination of tables with out nulls.

where union all retuns combination of two tables with duplicates

create table employee\_dup

(

  id int,

  emp\_name varchar(100),

  gender varchar(100),

  salary int,

  dept\_id int,

  location\_id int

 )

 insert into employee\_dup

 select \* from employee where dept\_id not in(1,2,3,4)

 create table employee\_dup2

(

  id int,

  emp\_name varchar(100),

  gender varchar(100),

  salary int,

  dept\_id int,

  location\_id int

 )

 insert into employee\_dup2

 select \* from employee where dept\_id not in(5,3,8,9)

 --union

select \* from employee

union

select \* from employee\_dup

union

select \* from employee\_dup2

--union all

select \* from employee

union all

select \* from employee\_dup

union

select \* from employee\_dup2

22)intersect

--returns comman records from both tables with out duplicates

select \* from employee

intersect

select \* from employee\_dup

intersect

select \* from employee\_dup2

23)except

--returns recordes in a which are not in b no duplicates in result.

insert into employee values(1,'employee 1','female',62000,4,5)

select \* from employee

except

select \* from employee\_dup

except

select \* from employee\_dup2

24)intersect vs in, except vs not in

--for all intersect, union,except all tables should have same number of columns and in same position.

--except vs not in

select \* from employee

except

select \* from employee\_dup

select \* from employee where id not in(select id from employee\_dup)

--we can do not in based on single column may retun duplicates and except can perform on multiple tables but not in based on two tables.

--intersect vs in

select \* from employee

intersect

select \* from employee\_dup

intersect

select \* from employee\_dup2

select \* from employee where id in(select id from employee\_dup)

select e1.\* from employee e1 inner join employee\_dup e2 on e1.id=e2.id inner join employee\_dup2 e3 on e2.id=e3.id

--we can do in based on single column may retun duplicates and except can perform on multiple tables but not in based on two tables.

--inner join can do on mutiple tables but based on single column retuns duplicates.

25)join

joins are used to join two tables based on columns in that columns

select \* from employee e join deptartment d on e.dept\_id=d.dept\_id --inner join gives matching from both tables

select \* from employee e left join deptartment d on e.dept\_id=d.dept\_id --left join gives all from left and matching from right[gives nulls for non matching].

select \* from employee e right join deptartment d on e.dept\_id=d.dept\_id --right join gives all from left and matching from right[gives nulls for non matching].

select \* from employee e full join department d on e.dept\_id=d.dept\_id --gives all matching and not matching from both tables.

select \* from employee e cross join department d --all possible matches between both tables.(250 in employee \* 13 in departments 3250 rows)

23)list out tables in database

select \* from sys.tables

select \* from sys.objects where type='u'

select \* from information\_schema.tables where table\_type='base table'

24)make sql script re runnable

--using object\_id()

if (object\_id('employee') is null)

begin

    create table employee

    (

    id int,

    name varchar(100),

    gender varchar(10)

    )

end

else

begin

    print 'table is already created'

end

--exists with sys.objects

if exists(select \* from sys.objects where type='u' and name='employee')

begin

    print 'table is already created'

end

else

begin

    create table employee

    (

    id int,

    name varchar(100),

    gender varchar(10)

    )

end

--exists with sys.tables

if exists(select \* from sys.objects where name='employee')

begin

    print 'table is already created'

end

else

begin

    create table employee

    (

    id int,

    name varchar(100),

    gender varchar(10)

    )

end

--exists with information.schema.tables

if exists(select \* from information\_schema.tables where table\_type='base table' and table\_name='employee')

begin

    print 'table is already created'

end

else

begin

    create table employee

    (

    id int,

    name varchar(100),

    gender varchar(10)

    )

end

--for column

if exists(select \* from sys.columns where name='gender' and [object\_id]=object\_id('employee'))

begin

    print 'column is already created'

end

else

begin

    alter table employee add gender varchar(100) not null

end

go

----

25)stored procedure it is a block of code which is pre compiled and stored in database. we can call it when we needed by following way

execute [procedure\_name]

exec [procedure\_name]

[procedure\_name]

Go

--basic stored procedure

create proc dept\_wise\_employee\_count

as

begin

    select dept\_id,count(\*) as emp\_count from employee group by dept\_id

end

exec dept\_wise\_employee\_count

go

--stored procedure with input parameter

create proc dept\_wise\_employees (@dept\_id as int)

as

begin

    select \* from employee where dept\_id=@dept\_id

end

declare @dept as int=2

exec dept\_wise\_employees @dept

exec dept\_wise\_employees 5  --we can pass values directly or by variable

go

--stored procedure with output parameter

create proc loaction\_wise\_dept\_wise\_count(@dept\_id as int ,@loaction\_id as int,@count as int output)

as

begin

    select @count=count(\*) from employee where dept\_id=@dept\_id and location\_id=@loaction\_id

end

declare @dept as int,@location as int --declaring mutiple variables at once

set @dept=4

set @location=1

declare @coun as int

exec loaction\_wise\_dept\_wise\_count @dept,@location,@coun out --out and output both are same

select @coun as 'employee\_count' --need to specify out or output for output variable.

Go

--stored procedure with mutiple output parameters for input parameters we no need to specify input but for output parameters we need to specify out ot output.

create proc loaction\_wise\_dept\_wise\_count\_sum\_min\_max\_avg\_salary

(@dept\_id as int ,@loaction\_id as int,@count as int out,@sum\_sal as int out,@min\_sal as int out, @max\_sal as int out, @avg\_sal as int out)

As

Begin

    Select

@count=count(\*),@sum\_sal=sum(salary),@min\_sal=min(salary),@max\_sal=max(salary),@avg\_sal=avg(salary)

    from employee where dept\_id=@dept\_id and location\_id=@loaction\_id

end

declare @dept as int,@location as int --declaring mutiple variables at once

set @dept=4

set @location=1

declare @coun as int,@sum as int,@min as int,@max as int, @avg as int

exec loaction\_wise\_dept\_wise\_count\_sum\_min\_max\_avg\_salary @dept,@location,@coun out,@sum out,@min out,@max out,@avg out--out and output both are same

select @dept as dept\_id,@location as loaction\_id,@coun as employee\_count,@sum as sum\_salary,@min as min\_salary,@max as max\_salary,@avg as avg\_salary

go

--use return in procedure

create proc return\_sum\_salary(@dept\_id as int)

as

begin

    return(select sum(salary) from employee where dept\_id=@dept\_id)

end

declare @dept as int=1

declare @sum as int

exec @sum=return\_sum\_salary @dept

select @sum as sum\_salary

--we cannot return mutiple columns using return use variables or a basic procedure as shown above to achieve result.

Go

--making parameters default

create proc default\_parameters (@dept\_id as int=null,@location\_id as int=null,@salary as int=null)

as

begin

    select \* from employee where (dept\_id=@dept\_id or @dept\_id is null) and (location\_id=@location\_id or

@location\_id is null)

    and (salary>=@salary or @salary is null)

end

declare @dept as int=1

declare @location as int=2

declare @sal as int

exec default\_parameters @dept,@location --here we are not passing the salary parameter thrm also we getting

output.

--to change procedure use alter in place of create.

Go

26)replace of nulls

--we can replace nulls by 3 methods using isnull,coalesce,case statment

--isnull if specified value is null it returns alternative value we specified.

select isnull(null,'pavan') as [name]

select isnull(null,null) as [name] --if alternative value also null it return null.

---coalesce it retun first not null value in all specified values we can specify n number of values

select coalesce(null,null,null,'pavan','kumar',null) as [name]

select coalesce(null,null,null,null,null,null) as [name] --if all vaues are null it returns error

--case statement

declare @name as varchar(100)=null

select case when @name is null then 'pavan' else null end as [name]

select case when @name is null then null else null end as [name] --give error both results not to be null.

27)create table type

--by using table type we can create table parameters actually parameters holds single value of certain data type but by table type parameter it can store a table.

create type [table\_type] as table

(

id int,

name varchar(100),

salary int,

gender varchar(100)

)

go

create proc table\_type (@table as [table\_type] readonly)as

begin

    select \* from @table

end

declare @table1 as [table\_type]

insert into @table1 select id,emp\_name,salary,gender from employee

exec table\_type @table1

--we need to pass table parameter as read only in stored procedure.

28)temporary tables.

/\*there are two types of trmport tables local and global

local represented by # and can accesed with the session it created.

global temporary represented by ## accesed in other sessions untill the session in which it created is present.\*/

create table #temp\_local

(

id int,

name varchar(10)

)

create table ##temp\_global

(

id int,

name varchar(100)

)

29)dynamic sql

dynamic sql helps to chnage the sql query based on user input.

in dynamic sql we write query as string and by using execute we execute that sql query.

declare @dynamicsql as varchar(100)

set @dynamicsql='select \* from employee'

execute(@dynamicsql)

go

--using dynamic sql in procedure

alter proc dynamic\_sql(@gtype as varchar(10),@agg\_type as varchar(10))

as

begin

    declare @func as varchar(100)

    declare @agg as varchar(100)

    declare @dynamicsql varchar(max)

    if @gtype in('d','g','l') and @agg\_type in ('s','a','mi','ma','c')

    begin

        if @gtype='d'

        begin

            set @func=' dept\_id '

        end

        else if @gtype='l'

        begin

            set @func=' location\_id '

        end

        else

        begin

            set @func=' gender '

        end

        if @agg\_type='s'

        begin

            set @agg=' sum(salary) as sum\_salary '

        end

        else if @agg\_type='a'

        begin

            set @agg=' avg(salary) as avg\_salary '

        end

        else if @agg\_type='c'

        begin

            set @agg=' count(salary) as count\_salary '

        end

        else if @agg\_type='mi'

        begin

            set @agg=' min(salary) as min\_salary '

        end

        else

        begin

            set @agg=' max(salary) as max\_salary '

        end

        set @dynamicsql='select '+@func+','+@agg+' from employee group by '+@func

        select @dynamicsql as 'query'

        execute (@dynamicsql)

    end

    else

    begin

        select 'please given correct input'

    end

end

exec dynamic\_sql 'd','s'

30)quotename and parsename

quote name wraps content in square brackets, single codes or duble codes etc to avoid sql injections.

sql injections mostly happens in dynamic sql dur to including unwanted text in it.

create  table dup

(

id int

)

create database d

declare @tablename as nvarchar(100)='[dup] drop database d'

declare @dynamicsql as nvarchar(100)='select \* form '+@tablename --it delets master database also it is called sql injection

print @dynamicsql

execute sp\_executesql @dynamicsql --to avoid that sql inject we use quotename with wraps text in single, double or square brackets defaut square brackets

declare @dynamicsql2 as nvarchar(100)='select \* from '+quotename('dbo')+'.'+quotename(@tablename)

print @dynamicsql2

select quotename(@tablename,'''') --enclosed in single codes

select quotename(@tablename,'""') --enclosed in single codes

select quotename(@tablename,'[]') --enclosed in single codes

parsename used to retun the particular part of given object.

select parsename([object\_name],[number])

--1 retuns object name

--2-returns schema name

--3-retuns databse name

--4-retuns server name

31)acid properties

acid properties are the properties must possess by transaction.

atomicity all commands in the transaction must run successfully if any one is not executed successfully all should rollback

consistent all data touched by the transaction must ne consistant

isolation each transaction must be isolated by other transaction.

durability if any change happen that must be permemenet.

32)select into and insert into

select id,emp\_name,gender into emp\_new from employee --to copy data from one table to other table which will created at run time both tables have same schema.

select id,emp\_name,gender into emp\_new from employee where 1==2 --to copy the schema of table.

insert into copy data from a table to new table which is already created.

insert into employee\_dup

select \* from employee

33)string functions

string functions used to work on strings.

select char(65) as alpha1,char(90) as alpha2 --convert given ascii number to corresponding character

select ascii('a') as [ascii] --convert character into ascii values

select upper('a') as [upper] --convert to upper case

select lower('a') as[lower] --convert to lower case

select trim('   pavan   ') as [trim] --removes trailing and leading spaces

select ltrim('   pavan  ') as [ltrim]  --removes left trailing spaces

select rtrim('   pavan  ') as [rtrim]  --removes right trailing spaces

select right('pavan',2) as [right] --gives specified characters from right side'

select left('pavan',2) as [left] --gives specified characters from left side'

select substring('pavan kumar thota',1,8) as [sub string] --gives a part of string from start to end [start and end included] specified.

select charindex('@','pavan kumar@157999@gmail.com') as [char\_index] --gives index of character specified at first occurance

select charindex('%','pavan kumar@157999gmail.com') as [char\_index] --gives index of character specified if not found return 0.

select patindex('%@%','pavankumar@157999@.com') as [pat\_index] --gives start index of given pattren at first match

select len('pavan kumar ') as [len] --gives length of given string

select replace('pavankumar15799@gmail.com','.com','.net') as [replace] --it replace old string with new string in the given string

select translate('pavankumar15799@gmail.com','.com','.net') as [replace] --it replace character to character in given string

select stuff('pavankumar',1,3,'\*\*\*\*\*\*\*\*\*') as [stuff] --it repalce string from start to end index[start and end included] with specified one.

select space(5) as [space] --it return sepecified number od spaces

select [value] from string\_split('pavan,kumar,thota',',') --it split string with specified delimieter and returns as column[value] in table

select dept\_id,string\_agg(emp\_name,'/') from employee group by dept\_id --it combines string with specified delimiter.

select concat('pavan',' ','kumar','thota') as [concat] --it concat given string

select concat\_ws('/',' ','kumar','thota','pavan') as [concat] --it concat given string with specified delimiter, delimeter specified at starting.

select reverse('pavan kumar thota') as [reverse] --it reverse the given string

34)dml

/\*dml data manupilation language

this are used to manipulated data

insert, delete, update comes under dml.\*/

--insert used to insert the records

insert into employee\_dup values(251,'employee-251','male',30000,3,2) --to insert all values into a table position must be same as columns in table.

insert into employee\_dup(id,emp\_name,gender,salary,dept\_id,location\_id) values(251,'employee-251','male',30000,3,2) --to insert all values into a table both are same

insert into employee\_dup(id,emp\_name,gender,salary,dept\_id) values(251,'employee-251','male',30000,3) --to specified values values into a table both are same

insert into employee\_dup(emp\_name,id,gender,salary,dept\_id) values('employee-251',251,'male',30000,3) --to specified values values we can insert values in any order into table.

alter table employee\_dup add constraint dk12 default 2 for location\_id

insert into employee\_dup(emp\_name,id,gender,salary,dept\_id,location\_id) values('employee-251',252,'male',30000,3,null) --to specified values values we can insert values in any order into table.

--even we give default if we pass null null will reflects.

--one column has default we want insert values in to other columns then also we need to give column names.

insert into employee\_dup values(251,'employee-251','male',30000,3,2),(251,'employee-251','male',30000,3,2) --insert multiple rows at a time.

--delete

--it used to delete records from the table here we can use where clause if we wont specify where clause it removes all records in table.

delete from employee\_dup --delete all records in employee\_dup

delete from employee\_dup where id=251

--update

--it used to update existing records in table we can use whee clause to filter which records to update and we can update multile columns at same time.

update employee\_dup set dept\_id=2 --it changes every employee dept\_id to 2

update employee\_dup set dept\_id=2 where det\_id=1  --update a column based on same colun value

update employee\_dup set dept\_id=3 where location\_id=3 --update one column based on value in other column.

update employee\_dup set dept\_id=2,location\_id=2 where emp\_name='employee 22' --updating muile columns.

35)iif

--iif is same as if we can called it as short hand if but else of flase result is mandatory.

--iif(condition,true,false)  flase part must if we give only true part it returns erro.

declare @name as varchar(100)='pavan'

select iif(@name='pavan','yes','no') as [name]

select iif(@name='pavan','yes') as [name] --return error

36)choose

--based on value given it select correspoinding value at that index.

select choose(month(getdate()),'jan','feb','mar','apr','may') as [month\_name]

select choose(10,'jan','feb','mar','apr','may') as [month\_name] --returns null

--if value at secefied index is not there it return null

37)date time functions

--date time function used to work on date and time.

select getdate() as [date] --it returns date and time presision 3

select sysdatetime() as [date] --it return date and time presision 7

select current\_timestamp as [date] --it samemas get date where get date is t-sql current \_time stamp is ansi sql.

select getutcdate() as [date] --it same as getdate() but it returns time at green witch mean time

select sysutcdatetime() as [date] --it same as sysutcdatetime() it return time at green witch mean time.

select sysdatetimeoffset() as [date\_offset]--it retuns offset with date and time

select month(getdate()) as [month] --it retuns month of given date

select month(123) as [month] --it returns sum of values

select day(getdate()) as [day] --it retun day of given date

select day(123) as [day] --it consider as day of year and return day number in that month.

select year(getdate()) as [year] --it retun year of given date

select year(123) as [year] --it return 1900 as output

--for year,day,month if we gave string they return error.

select isdate(getdate()) as [date] --it retun 1 if specified one is date

select isdate(sysdatetime()) as [date] --if we give datetime2 it retun error

select isdate('2ndhd') as [date] --it retuns 0 if specified in not date.

select eomonth(getdate()) as[eomonth] --it return end date of current month

select eomonth(getdate(),-1) as[eomonth] --it return end date of previous month - for previous

select eomonth(getdate(),1) as[eomonth] --it return end date of next month + for next

day-day of the month

dayofyear of the year

week-week of year

weekday-day of week

month-month of year

year-year from date

quarter-quarter of year

hour-hour of day

minute-minute of hour

second-second of minute

millisecond-millisecond of second

tzoffset-returns offset of time

select datepart(day,getdate()) as [day]

select datepart(dayofyear,getdate()) as [dayofyear]

select datepart(week,getdate()) as [weekofyear]

select datepart(weekday,getdate()) as [dayofweek]

select datepart(month,getdate()) as [month]

select datepart(quarter,getdate()) as [quarter]

select datepart(year,getdate()) as [year]

select datepart(hour,getdate()) as [hour]

select datepart(minute,getdate()) as [minute]

select datepart(second,getdate()) as [second]

select datepart(millisecond,getdate()) as [milli seconds]

select datepart(tzoffset,sysdatetimeoffset()) as [tzoffset] --return in minutes

select right(cast(sysdatetimeoffset() as varchar(100)),6) as [tzoffset] --retuns tzoffset

select datename(day,getdate()) as [day]

select datename(dayofyear,getdate()) as [dayofyear]

select datename(week,getdate()) as [weekofyear]

select datename(weekday,getdate()) as [day\_name\_in\_week]

select datename(month,getdate()) as [month\_name]

select datename(quarter,getdate()) as [quarter]

select datename(year,getdate()) as [year]

select datename(hour,getdate()) as [hour]

select datename(minute,getdate()) as [minute]

select datename(second,getdate()) as [second]

select datename(millisecond,getdate()) as [milli seconds]

select datename(tzoffset,sysdatetimeoffset()) as [tzoffset] --return tzoffset

select right(cast(sysdatetimeoffset() as varchar(100)),6) as [tzoffset] --retuns tzoffset

--note:-main difference between datapart and datename is datepart return int values for month,weekday,tzoffset where datename returns string values.

select dateadd(day,1,getdate()) as [day]

select dateadd(week,1,getdate()) as [week]

select dateadd(dayofyear,1,getdate()) as [dayofyear]

select dateadd(weekday,1,getdate()) as [weekday]

select dateadd(month,1,getdate()) as [month]

select dateadd(quarter,1,getdate()) as [quarter]

select dateadd(year,1,getdate()) as [year]

select dateadd(hour,1,getdate()) as [hour]

select dateadd(minute,1,getdate()) as [minute]

select dateadd(second,1,getdate()) as [second]

select dateadd(millisecond,1,getdate()) as [millisecond]

--tzoffset is not suported in dateadd and day,weekday,dayofweek adds day to given date dateadd add given value with respective to intervals.

select datediff(day,'2022-03-03 22:10:23:234',getdate()) as [day]

select datediff(weekday,'2022-03-03 22:10:23:234',getdate()) as [weekday]

select datediff(week,'2022-03-03 22:10:23:234',getdate()) as [week]

select datediff(month,'2022-03-03 22:10:23:234',getdate()) as [month]

select datediff(quarter,'2022-03-03 22:10:23:234',getdate()) as [quarter]

select datediff(year,'2022-03-03 22:10:23:234',getdate()) as [year]

select datediff(hour,'2022-03-03 22:10:23:234',getdate()) as [hour]

select datediff(minute,'2022-03-03 22:10:23:234',getdate()) as [minute]

select datediff(second,'2022-03-03 22:10:23:234',getdate()) as [second]

select datediff(millisecond,'2022-03-03 22:10:23:234',getdate()) as [millisecond] --over flow to large value for small value it gives result

--dayofyear,tzoffset are not supported with datediff datediff return difference between two dates in specified intervals.

select datefromparts(2019,02,03) as [date] --convert parts to date

select datefromparts(2019,02,null) as [date] --retuns null if any parameter is null

select datefromparts(2019,02,'0h3') as [date] --it try to convert string to int if it converable ('03') gives result other wise error.

select smalldatetimefromparts(2019,02,03,03,30) as [date\_time] --convert parts to datetime

select smalldatetimefromparts(2019,02,null,03,30) as [date\_time] --retuns null if any parameter is null

select smalldatetimefromparts(2019,02,'0h3',03,30) as [date\_time] --it try to convert string to int if it converable gives result other wise error.

select timefromparts(23,23,23,230,3) as [time] --last one is presision it must be greater than len(millisecond)

select timefromparts(23,23,23,230,null) as [time]  --null to presision gives error

select timefromparts(23,23,null,230,3) as [time] --retun null if any one parameter is null except presision

select timefromparts(23,23,'s23',230,3) as [time]  --it try to convert string to int if it converable gives result other wise error.

select datetimefromparts(2019,02,03,03,30,23,230) as [date\_time] --convert parts to datetime

select datetimefromparts(2019,02,null,03,30,20,230) as [date\_time] --retuns null if any parameter is null

select datetimefromparts(2019,02,'0h3',03,30,20,230) as [date\_time] --it try to convert string to int if it converable gives result other wise error.

select datetime2fromparts(2019,02,03,03,30,23,230,4) as [date\_time] --convert parts to datetime

select datetime2fromparts(2019,02,'03',03,30,20,230,null) as [date\_time] --retuen error if presision is null

select datetime2fromparts(2019,02,null,03,30,20,230,4) as [date\_time] --retuns null if any parameter is null except presision because presision gives error.

select datetime2fromparts(2019,02,'0h3',03,30,20,230,6) as [date\_time] --it try to convert string to int if it converable gives result other wise error.

--datetime-1753-9999,3.33 milli seconds accuracy,8 bytes

--date-day accuracy

--smalldatetime-1900-2079, minute accuracy, 4 bytes

--datetime2-0001-9999,100 nano seconds accuracy, presison 1,2-4 bytes, 3,4-5 bytes, 5,6,7-6 bytes

38)temp table in dynamic sql

create proc dyn

as

begin

    declare @dynamicsql as varchar(max)

    set @dynamicsql='create table #temp(id int)

                    insert  into #temp values (1)

                    select \* from #temp'

    execute(@dynamicsql)

end

exec dyn

alter proc dyn

as

begin

    declare @dynamicsql as varchar(max)

    set @dynamicsql='create table #temp(id int)

                    insert  into #temp values (1)'

    execute(@dynamicsql)

    select \* from #temp

end

exec dyn

alter proc dyn

as

begin

    create table #temp(id int)

    declare @dynamicsql as varchar(max)

    set @dynamicsql='insert  into #temp values (1)

                    select \* from #temp'

    execute(@dynamicsql)

end

exec dyn

alter proc dyn

as

begin

create table #temp(id int)

    declare @dynamicsql as varchar(max)

    set @dynamicsql='insert  into #temp values (1)'

    execute(@dynamicsql)

    select \* from #temp

end

exec dyn

--tables create outside dynamic block can accesed by dynamic block

--tables created in dynamic block cannot accessd at outside.

39)keys in dbms

there are different types of keys in dbms

super key-all possible combination of columns or attributes used to uniqely identify the records in a table called super key.

candidate key-the minimal sets of keys that can unqely identify the records in a table is candidate key. it may have nulls.

primary key-primary key is a column or group of columns used to identify records in the uniquely.id does not contain nulls and duplicates.

foreign key-foreign key is a column or group of columns that refers primary key in other table.

composite key- if primary key formed by multiple columns then it called compostie key.

alternate key- other keys than primary key in candidate key is nothing but alternative key.

artificial or surogate key- new column created to use it a primary key mostly a numeric column with identity.

primary key is subset of candiadte key, candiadte key is subset of super key.

aternate key=candidate key-primary key.

40)cross apply and outer apply.

/\*cross apply and outer apply used to perform join between table or view with function

cross apply same as inner join and outer apply same as left outer join.\*/

select \* from employee e cross apply dbo.dept(e.dept\_id)

select \* from employee e outer apply dbo.dept(e.dept\_id)

41)sequence object

sequence object is same as identity it gives incremnet value but identity confined to a table where we can use sequence across tables

and we can specify max value, min value, incremnet cache etc.

create sequence dbo.seq as int

start with 1 --starting value

increment by 1 --increment value

minvalue 1 --min value must be greater that start

maxvalue 1000 --max value

cycle --cycle restarts after reaching max value noncycle not reserts after max value

cache 10 --it stores next coming 10 values.

select next value for dbo.seq

42)cursor

--cursor used to iterate table row by row they are very slow try to avoid them as much as possible.

declare cur cursor for select id,emp\_name,dept\_id,location\_id from employee --create cursor

open cur --to open cursor

declare @id as int

declare @dept\_id as int

declare @location\_id as int

declare @emp\_name as varchar(100)

fetch next from cur into @id,@emp\_name,@dept\_id,@location\_id

while (@@fetch\_status=0)

begin

    select @id,@emp\_name,@dept\_id,@location\_id

    fetch next from cur into @id,@emp\_name,@dept\_id,@location\_id

end

close cur --to close cursor

deallocate cur --to delete cursor

--when no row to fetch @@fetch\_status become 1

43)index

index are used to speed up the query

there are two types on indexs clustered and non clustered index we can give unique property can assign to bith indexs.

clustered index is the index which define the physical order of table and it stored in the table, clustered index is fast compared to non clustered index.

each table only have one clustered index.

unique clustered index is nothng but a primary key when ever we create primary key back side a unique clustered index is created.

create unique clustered index u\_c\_i on employee (id asc)

non clustered index are used to speed up the query we may have mutiple non clustered index create non clustered index on most used columns because

they stored in heap it increases memory consumption we can have up to 999 non clusterd index.

create nonclustered index nu\_i\_c on employee(emp\_name desc)

--first we need to create clustered index before creating non cluster index.

44)mathematical functions

--used to perform mathematical functions

select abs(-23) as [abs1], abs(23) as [abs2] --it return positive value wether value is positive or negitive

select square(2) as [square] --it return square of given value

select power(2,3) as [power] --it return value by doing power of given values.

select sqrt(4) as [sqrt] --it return square root of given value

select floor(2.90) as [floor] -- it return round down value of given value

select ceiling(2.1) as [ceiling] --it return round up value of given value.

select round(234.342,0) as [round] --it truncates decimals

select round(234.342,1) as [round] --it truncates decimal to one decimal place

select round(234.342,-1) as [round] --it truncates up to one values before decimal value

select rand() as [rand] --it retun rand value between 0[inclusive] and 1[exlusive]

select rand(1) as [rand\_seed] --it return same values between 0[inclusive] and 1[exclusive]

45)row\_number(),rank(),dense\_rank()

/\*row\_number generates numbers in sequence for each row.1,2,3,4,5

rank gives rank to records based on articular column and partion (optional) it skip rank to next rows if before two rows have same rank.1,2,2,4,5

dense\_rank is same as rank but dense rank does not skip rank to next record if two records have same rank.1,2,2,3,4\*/

select \*,row\_number() over (order by salary desc) as [row\_number] from employee --give rownumber order by salary desc

select \*,row\_number() over (partition by dept\_id  order by salary desc) as [row\_number] from employee ----give rownumber order by salary desc and partition by dept\_id

select \*,rank() over (order by salary desc) as [row\_number] from employee --give rank order by salary desc

select \*,rank() over (partition by dept\_id order by salary desc) as [row\_number] from employee--give rank order by salary desc and partition by dept\_id

select \*,dense\_rank() over (order by salary desc) as [row\_number] from employee --give demse\_rank order by salary desc

select \*,row\_number() over (partition by dept\_id order by salary desc) as [row\_number] from employee --give dense\_rank order by salary desc and partition by dept\_id

46)guid

/\*guid means global unique identifier which is unique across databse and server.

we can generate guid with newid() function. datatype of guid is unique identifier.

it is 16 bytes length\*/

declare @guid as uniqueidentifier

if @guid is null

begin

    set @guid=newid()

end

print @guid

declare @empty\_guid as uniqueidentifier

set @empty\_guid=cast(0x0 as uniqueidentifier)

print @empty\_guid

--empty guid is a guid which consist of all zeroes.

47)merge

--erge used to convert target table into source table i.e one table to other both tables should have same metadata.

select \* from employee

select \* from employee\_dup

--before we dooing merge we need to ensure that both tables does not have duplicates on column on which we using to match both tables if duplicates are there

--delete them using cte.

merge [employee\_dup] as t

using [employee] as s

on t.id=s.id

when matched then

update set t.id=s.id,t.emp\_name=s.emp\_name,t.gender=s.gender,t.salary=s.salary,t.dept\_id=s.dept\_id,t.location\_id=s.location\_id

when not matched by target then

insert  values(s.id,s.emp\_name,s.gender,s.salary,s.dept\_id,s.location\_id)

when not matched by source then

delete;

48)rollup,grouping sets, cube

--above 3 are used to do muti level grouping

--grouping sets group based on given sets.

--rollup groups based on hiearchy

--cube groups in all possible ways.

--grouping sets

select isnull(cast(dept\_id as varchar(10)),'dept\_wise') as dept\_id,

isnull(cast(location\_id as varchar(10)),'location\_wise')  as location\_id,isnull(gender,'gender\_wise') as gender,

isnull(sum(salary),0) as [salary] from employee

group by grouping sets

(

(dept\_id,location\_id,gender),

(dept\_id,location\_id),

(dept\_id),

()

)

--rollup

select isnull(cast(dept\_id as varchar(10)),'dept\_wise') as dept\_id,

isnull(cast(location\_id as varchar(10)),'location\_wise')  as location\_id,isnull(gender,'gender\_wise') as gender,

isnull(sum(salary),0) as [salary] from employee

group by rollup(dept\_id,location\_id,gender)

select isnull(cast(dept\_id as varchar(10)),'dept\_wise') as dept\_id,

isnull(cast(location\_id as varchar(10)),'location\_wise')  as location\_id,isnull(gender,'gender\_wise') as gender,

isnull(sum(salary),0) as [salary] from employee

group by dept\_id,location\_id,gender with rollup

--both are same

--cube

select isnull(cast(dept\_id as varchar(10)),'dept\_wise') as dept\_id,

isnull(cast(location\_id as varchar(10)),'location\_wise')  as location\_id,isnull(gender,'gender\_wise') as gender,

isnull(sum(salary),0) as [salary] from employee

group by cube(dept\_id,location\_id,gender)

select isnull(cast(dept\_id as varchar(10)),'dept\_wise') as dept\_id,

isnull(cast(location\_id as varchar(10)),'location\_wise')  as location\_id,isnull(gender,'gender\_wise') as gender,

isnull(sum(salary),0) as [salary] from employee

group by dept\_id,location\_id,gender with cube

49)grouping, grouping\_id

--grouping and grouping\_id function sused to find which columns are in grouping. where in grouing it takes one column at a time and return 0 if not grouped and return

--1if it grouped. in gruping\_id takes mutiple columns and return result in decimal form we need to convert into binary to understand which column is in grouping

--eg we get 4 means 100 i.e first column is grouped remmining two column not grouped.

select isnull(cast(dept\_id as varchar(10)),'dept\_wise') as dept\_id,

isnull(cast(location\_id as varchar(10)),'location\_wise')  as location\_id,isnull(gender,'gender\_wise') as gender,

isnull(sum(salary),0) as [salary],

grouping(dept\_id) as g\_d, grouping(location\_id) as g\_l,grouping(gender) as g\_g,grouping\_id(dept\_id,location\_id,gender) as g\_dlg

from employee

group by dept\_id,location\_id,gender with cube

50)sp\_helptext

--sp\_helptext used to return text written for the object. to avoid return the text written for object use with encryption as shown below.

create procedure sp\_help\_text

as

begin

    select \* from employee

end

sp\_helptext sp\_help\_text

alter procedure sp\_help\_text

with encryption

as

begin

    select \* from employee

end

sp\_helptext sp\_help\_text

51)sp\_rename

sp\_rename used to rename the table or column in the table

to reamee table sp\_rename 'old\_table\_name','new\_table\_name'

to rename column sp\_rename 'table\_name.old\_column\_name','new\_column\_name'

sp\_rename 'emp','emps'

sp\_rename 'emps.id','e\_id'

52)sp\_depends

--sp\_depends used to return the object refered or refering the object we mentioned.

sp\_depends 'employee' --if we pass refered entity it return referencing objects

sp\_depends 'dept'  --if we pass referencing entity it retun refered entity with columns on which it refering.

53)user defined function

functions are two types one user defined function and build in function

build in function are functions which are defined by sql server it self.

user defined functions are functions which are defined by us based on requirement.

there are 3 types of user defined functions

scalar value function which return a scalar value that value must not be image,ntext, cursor

inline table valued function return a table output

multi statement table valued function returns a table of defined type as output.

--scalr valued function

create function no\_of\_days(@date as date)

returns int

as

begin

    declare @days as int

    select @days=datediff(day,@date,getdate())

    return @days

end

select dbo.no\_of\_days('2022-02-02') as 'days'

--inline table valued function

create function locations(@location\_id as int)

returns table

as

return(select \* from employee where location\_id=@location\_id)

select \* from locations(1)

--multi statment table valued function

create function dept(@dept\_id int)

returns @table table(id int,name varchar(100),salary int,dept\_id int,location\_id int)

as

begin

    insert into @table

    select id,emp\_name,salary,dept\_id,location\_id from employee where dept\_id=@dept\_id

    return

end

select \* from dept(1)

--function which give same output with same set of input is called deterministic function. sum,avg etc.

--function which gives different output at each time it executes is called non deterministic function. rand()

--rand(1) is a deterministic function it return same value each time we execute. where rand(0 gives different output.

54)triggers

triggres are special type of stored procedures which executes when ever an event occurs in the database or server.

there are 3 types of triggers

dml - delete, insert, update each of the divided in to after insert and istead of insert.

ddl -create,alter, drop

logon - to control over logins.

dml- datamanupulation triggers this triggers are trigger when dml commands like insert delete, update executed on table.

when we insert new record values are stored in inserted when we delete record it stored in deleted this inserted and deleted tables are avilible with in trigger only

this tables are called magic tables.

usually triggers are triggered once per batch not for each row.

create table emp

(

id int,

name varchar(100),

salary int,

gender varchar(100)

)

--after insert trigger, trigger triggers after inserting data into table.

create table after\_insert

(

id int identity(1,1),

record varchar(100)

)

create trigger after\_insert\_emp on emp for insert

as

begin

    declare @id as int

    declare @name as varchar(100)

    declare @salary as int

    declare @gender as varchar(100)

    select @id=id,@name=name,@salary=salary,@gender=gender from inserted

    insert into after\_insert values(cast(@id as varchar(100))+' '+@name+' '+cast(@salary as varchar(100))+' '+@gender)

end

insert into emp values (1,'pavan',20000,'m'),(2,'sasi',250000,'m'),(3,'babavali',300000,'f')

select \* from after\_insert

--after delete triggers after delete the data

create table after\_delete

(

id int identity(1,1),

record varchar(100)

)

create trigger after\_delete\_emp on emp for delete

as

begin

    declare @id as int

    declare @name as varchar(100)

    declare @salary as int

    declare @gender as varchar(100)

    select @id=id,@name=name,@salary=salary,@gender=gender from deleted

    insert into after\_delete values(cast(@id as varchar(100))+' '+@name+' '+cast(@salary as varchar(100))+' '+@gender)

end

select \* from after\_delete

delete from emp where id in (2,3)

--after update it triggers after updating data in table.

create table after\_update

(

id int identity(1,1),

new\_record varchar(100),

old\_record varchar(100)

)

alter trigger after\_update\_emp on emp for update

as

begin

    declare @id as int,@id2 as int

    declare @name as varchar(100),@name2 as varchar(100)

    declare @salary as int,@salary2 as int

    declare @gender as varchar(100),@gender2 as varchar(10)

    select @id2=id,@name2=name,@salary2=salary,@gender2=gender from inserted

    select @id=id,@name=name,@salary=salary,@gender=gender from deleted

    insert into after\_update(old\_record,new\_record) values(cast(@id as varchar(100))+' '+@name+' '+cast(@salary as varchar(100))+' '+@gender,

    cast(@id2 as varchar(100))+' '+@name2+' '+cast(@salary2 as varchar(100))+' '+@gender2)

end

select \* from after\_update

update emp set name='babavali' where name='bab'

update emp set name='pa' where id in (1,2)

--instead of insert instead of insert trigger will trigger

--with this we can validate data before we insert data like vale in certain limit and datatype etc.

go

55)cte

--cte means comman table expresion

--which used to stoe the temporary result set and we can access it right below it created.

--there are different types of ctes

select \* into #ed from employee

select \* into #dd from deptartment

select \* into #ld from location

go

--simple  cte with arameter names

with cte (id,name,sal)

as

(

    select id,emp\_name,salary from #ed

)

select \* from cte

--parameters just give alias name to columns comming from cte both columns from cte and number of parameters must be same.

go

--simple cte with out parameters

with cte

as

(

    select \* from #ed

)

select \* from cte

--every column in cte must have name to column (sum(salary) no name given ) if any column don't have name it return error.

--every column must repeat one time only.

go

--mutiple ctes

--we can write mutiple ctes with one with and each cte seperated by comma.

with cte

as

(

    select \* from #ed

),

cte2

as

(

    select \* from #dd

)

select \* from cte,cte2 --retuns carticien product

go

--multi level cte

--access above cte in next cte

with cte

as

(

    select \* from #ed

),

cte2

as

(

    select c.\*,d.dept\_head\_id,d.dept\_name from cte c join #dd d on c.dept\_id=d.dept\_id

),

cte3

as

(

    select c.\*,l.location\_head\_id,l.location\_name from cte2 c join #ld l on c.location\_id=l.location\_id

)

select \* from cte3

--we can access all ctes in muti level cte like cte,cte2,cte3.

go

--recursive cte

--recursive cte refers it selft untill the condition satisfy if we wont specify condition it goes to infinite looping like while.

--we can generate series of dates by recursive cte which is most popular example for recursive cte.

declare @stdate as date=getdate()

declare @enddate as date='2025-01-01';

with cte

as

(

    select @stdate as [date]

    union all

    select dateadd(day,1,[date]) as [date] from cte where dateadd(day,1,[date])<=@enddate

)

select \* from cte

option(maxrecursion 1000) --max recursions done by cte is 100 but here we need more than 100 so we use option(maxrecursion no\_of\_recursion)

go

--dml by cte

--we can do insert, update, delete on table by cte.

--cte depond on single table

with cte

as

(

    select \* from #ed

)

insert into cte values(251,'employee 251','male',23400,4,2)

delete from cte where id=251

update cte set salary=50000 where id in (1,2,3,4)

--when cte based on one table we do dml reflects correctly in table.

go

--cte depond on mutiple table but dml effects one table

with cte

as

(

    select e.\*,d.dept\_head\_id,d.dept\_name from #ed e join #dd d on e.dept\_id=d.dept\_id

)

insert into cte values(251,'employee 251','male',23400,4,2) --gives error

delete from cte where id=251 --gives error

update cte set salary=190000 where id in (1,2,3,4,5)--update works

select \* from #ed

select \* from #dd

--insert and delete not possible when cte deponds upon mutiple tables and dml eefects one table it works.

go

--cte depond on mutiple table and dml effects multiple tables

with cte

as

(

    select e.\*,d.dept\_head\_id,d.dept\_name from #ed e join #dd d on e.dept\_id=d.dept\_id

)

insert into cte values(251,'employee 251','male',23400,4,2,9,'dept 2') --gives error

delete from cte where dept\_head\_id=251 --gives error

update cte set dept\_id=23 where id=8 --updates but not the ways we needed.

--insert and delete not possible when cte deponds upon mutiple tables and dml eefects one table update works butif it effects mutiple

--tables it won't work as we want.

-

56)views

--views are virtual tables which does not store data they just store query. views cal also used as abraction to database objects

--every column in view must have name to column (sum(salary) no name given ) if any column don't have name it return error.

--every column must repeat one time only.

go

create view em

as

select \* from employee

select \* from em

--materalized view or index on view.

--when we craete index on view it become materialized and stores the data.

--we need to follow some rules while we create index on views

--table name must be two part name, don't use \*, use schemabinding

go

create view emp1

with schemabinding

as

select id,emp\_name,gender,salary from dbo.employee

create unique clustered index u\_c\_i\_1 on emp1(id asc)

--first we need to create clustered index before creating non cluster index.

create unique nonclustered index u\_c\_i\_2 on emp1(emp\_name asc)

--note:-go defines end of batch view,cte, procedure palce them at staring of batch i.e. before it use go to end previous batch and to be only statment in batch.

--note:-to alter trigger, procedures, sequnece, views use alter in place of create.

--schemadinding makes object bind with schema.

--disadvantages of view

--we unable to pass parameters

--we cannot create view on temp tables.

--dml on table by view.

--we can do dml on table by using views.

--view based on one table

select \* into ed from employee

select \* into dd from deptartment

go

create view dml1

as

select \* from ed

insert into dml1 values(251,'employee 251','male',23400,4,2)

delete from dml1 where id=251

update dml1 set salary=50000 where id in (1,2,3,4)

--dml works fine when view based on one table.

go

--view based on multiple tables and dml affects one table.

create view dml2

as

select e.\*,d.dept\_head\_id,d.dept\_name from ed e join dd d on e.dept\_id=d.dept\_id

go

insert into dml2 values(251,'employee 251','male',23400,4,2)

delete from dml2 where id=251

update dml2 set salary=150000 where id in (1,2,3,4,5,6,7)

--same as cte insert delete not works but update works.

go

--view based on multiple tables and effects multiple table

create view dml3

as

select e.\*,d.dept\_head\_id,d.dept\_name from ed e join dd d on e.dept\_id=d.dept\_id

go

insert into dml3 values(251,'employee 251','male',23400,4,2,12,'dept-2')

delete from dml3 where dept\_head\_id=251

update dml3 set salary=150000 where dept\_id in (1,2,3,4,5,6,7)

--same as cte insert delete not works but update works when we effects one table but when we effect mutiple tables it won't works as we want.

--note:-dml effect same in both cte and view.

57)pivot

--pivot used to convert distinct values in the column to seperate column.

--pivot used for table shaping.

--if table has only gender,salart,dept\_id it aggeregates data other wise it just convert male and female in gender to two sepearate columns and have all 250 columns.

select dept\_id,gender,salary into #j from employee

select dept\_id,isnull(male,0) as male,isnull(female,0) as female from #j

pivot

(

    sum(salary) for gender in ([male],[female])

) as pivot\_table

58)unpivot

--it convert columns into distinct values in a column.

--it also used for table shaping.

--unpivot reverse pivot when data is not aggregated in pivot.

select dept\_id,gender,salary from

(

select dept\_id,isnull(male,0) as male,isnull(female,0) as female from #j

pivot

(

    sum(salary) for gender in ([male],[female])

) as pivot\_table

) as d

unpivot

(

    salary for gender in ([male],[female])

) as un\_p

59)masking

create table mask

(

id int,

name varchar(100),

age int,

join\_date date,

email varchar(100),

cards varchar(100),

phone\_number varchar(100)

)

--deafult gives 0000 based on values for int, 1900-01-01 for date, xxx for int.

--random gives random value between secified values in random

--email display first character @ and .com use for email if we not give email also it shows same.

--partial used to display specified first and last characters and remming with specified pattren.

--must specify start and end in partial don't want any characters specify 0 at start and end

alter table mask alter column id int masked with(function='default()')

alter table mask alter column name varchar(100) masked with(function='default()')

alter table mask alter column join\_date date masked with(function='default()')

alter table mask alter column email varchar(100) masked with(function='email()')

alter table mask alter column age int masked with(function='random(0,1)')

alter table mask alter column cards varchar(100) masked with(function='partial(2,"xxxxxx",4)')

alter table mask alter column phone\_number varchar(100) masked with(function='partial(2,"xxx-xxx",4)')

alter table mask alter column cvv varchar(100) masked with(function='partial(0,"xxxx",0)')

--keep pattren in double codes.

insert into mask values (1,'pavan',23,getdate(),'pavankumar157@gmail.com','2345-2344-9871','909703485')

insert into mask values (1,'pavan',23,getdate(),'pavankuma','2345-2344-9871','909703485')

alter table mask add cvv varchar(100)

create user test without login  --to cretae login

grant select on schema::dbo to test --grand select access on dbo to test user

execute as user='test' --to execute as test user

select \* from mask

grant unmask to test --to unmask the user.

select \* from mask

drop user test --to drop user.

60)transaction

/\*transaction treats block of commands as single unit if any one command not executed then all commands are rolled back.

use begin transaction to start a transcation

to commit changes use commit

to reverse all chnages done in transaction use rollback transaction.

savepoints hold all chnages occured up to that point save transaction [name\_of\_save\_point].

to rollback to savepoint rollback transaction [name\_of\_save\_point].\*/

create table trans

(

id int,

name varchar(100),

gender varchar(10)

)

begin transaction

insert into trans values (1,'pavan','m')

insert into trans values (2,'sasi','m')

save transaction sp1

insert into trans values (3,'babavali','m')

insert into trans values (4,'murali','m')

save transaction sp2

insert into trans values (5,'ramki','m')

insert into trans values (6,'purna','m')

save transaction sp3

rollback transaction sp1

rollback transaction sp2

commit transaction

select \* from trans

--by using savepoint we can rollback to particular point of transaction.

--once we commit the transaction all save points are ersased.

set transaction isolation level read committed

61)ddl triggers

ddl trigger triggers for any ddl event in databse or server.

ddl means create,drop,alter

we can create trigger for sp\_rename also.

we cna create trigger for multiple events.

by using rollback, we can rollback any ddl on server and databse.

won't trigger for temp tables.

they don't effect select into, insert into

go

--trigger for create\_table

create trigger ddl1 on database for create\_table

as

begin

    print 'print table created'

end

create table t1(id int)

go

--avoid creating table

create trigger ddl2 on database for create\_table

as

begin

    rollback

    print 'print table created'

end

create table t2(id int)

go

--multiple triggers

create trigger ddl3 on database for create\_table,alter\_table,drop\_table

as

begin

    rollback

    print 'print table created'

end

go

--rename trigger

create trigger ddl4 on database for rename

as

begin

    rollback

    print 'print rename is not possible'

end

go

sp\_rename 'emps','k'

go

--ddl on server level

create trigger ddl5 on all server for create\_table

as

begin

    rollback

    print 'print table created'

end

create table f(id int)

go

disable trigger ddl5 on all server --to disable trigger

go

enable trigger ddl5 on all server --to enable trigger

--we can enable and disable trigger at there efined scope only.

62)retuen multiple columns from subquery

--to return multiple values from sub query use sub query in from clause. it is not possible to get from select clause.

select dept\_id,(select count(\*) from employee e where e.dept\_id=d.dept\_id) as head\_count,

(select sum(salary) from employee e where e.dept\_id=d.dept\_id) as salary from deptartment d

--we can do above by below query which is optimised.

select k.\* from (select dept\_id,sum(salary) as salary,count(\*) as head\_count from employee group by dept\_id) as k join deptartment d on k.dept\_id=d.dept\_id

63)improve select query performance

--use column names instead of \*

--avoid using distinct

--use wild cards at end like 'c%' not like '%c%' if possible.

--don't use cartition products select \* from employee e deptartment d where e.dept\_id=d.dept\_id it makes all combination with dept and emp and then filter in it rather use inner join.

--filter data in where clause rather than having clause

64)having clause

select \* from employee having salary>50000 --having is invalid

select \* from employee having sum(salary)>50000-- having invalid

--with out group by we cannot use having.

select id,dept\_id,salary from employee group by id,dept\_id,salary having salary>50000 -- we can use columns with out aggregation on having but it need to be in group by.

select dept\_id from employee group by dept\_id having sum(salary)>200000 --we can use column with aggregate function but group by clause is mandatory.

--column participate in having either aggregated and group by clause is necessary or mention in group by.

65)values

--it used to convsert values into table

select \* from (values(1,2,3),(2,3,4),(4,5,6)) as tempatory(a,b,c)

select \* from (values (v1,v2,v3),(v1,v2,v3)) as table\_name(c1,c2,c3)

66)Create Table Variable

Declare @table as table(id int, name varchar(100))

select \* from @table

table variable can be accessed with in the batch.

We can give table type to any variable in the database.

If we assign table type to variable its structure is fixed i.e. structure at time of created.

By table variable we can define structure at time of declare variable i.e. flexible structure.

67)Natural Join

Natural Join does not exists in SQL Server.

Natural Join means joining two tables having same column name with out specify the column names.

select \* from EMPLOYEE

natural join DEPARTMENT

68)Getting all rows from CTE/subquery with order by

If we use top 100 percent it return rows but not order the records.

So use offset/fetch next

declare @val as int

select @val=count(\*) from EMPLOYEE;

with cte as

(

  select  \* from EMPLOYEE

  order by salary desc

  offset 0 rows

  fetch next @val rows only

)

select \* from CTE

declare @val as int

select @val=count(\*) from EMPLOYEE

select \* from

(select \* from Employee order by salary desc

offset 0 rows fetch next @val rows only) as d

69)Row\_number without sorting any column.

select \*,

row\_number() over (order by (select 1)) as rn

from EMPLOYEE

70)Getting Float in result

declare @val1 as int=10

declare @val2 as int=3

select cast(@val1 as float)/@val2

select @val1/cast(@val2 as float)

select @val1\*1.0/@val2

select @val1/@val2\*1.0 –not return float

declare @val1 as float=10

declare @val2 as float=3

select @val1/@val2

declare @val1 as int=10

declare @val2 as float=3

select @val1/@val2

--if anyone value is float or cast as float it return float but for \*1.0 it return float it we \*1.0 with numerator only.

71)Age Function

Alter function Age(@date as date)

returns varchar(100)

AS

begin

    declare @return as varchar(100)=''

    if day(@date)<>day(getdate())

    BEGIN

        set @return=@return+cast(year(getdate())-year(@date)-1 as varchar(100))+' Years '

        set @return=@return+cast(12-abs(month(getdate())-month(@date)) as varchar(100))+' months '

        set @return=@return+cast(abs(day(getdate())-day(@date)) as varchar(100))+' days'

    End

    Else

    Begin

        set @return=cast(year(getdate())-year(@date)-1 as varchar(100))+' Years '

     End

     return @return

End

select dbo.Age('2000-09-15')

72)Set NoCount On

By using Set NoCount On it avoids getting number of rows aeffected by the query in message tab.

for examle we need to insert million of rows into table we get message 1 row effects millions of time i.e. for each row one

which reduces perfermance to print it at message tab. by kept set nocount on tgis message wonot print in message tab.

by default we have set nocount off.

set NoCount On

73)@@RowCount

it return number of rows effected by our query.

74)Inner Joins with where

Select \* from EMPLOYEE E,DEPTARTMENT D where E.DEPT\_ID=D.DEPT\_ID

except

Select \* from EMPLOYEE E join DEPTARTMENT D on E.DEPT\_ID=D.DEPT\_ID

75)Extract First,Middle and Last.

Declare @name as varchar(100)='Pavan Kumar Thota'

declare @spaces as int=len(@name)-len(replace(@name,' ',''))

select @spaces

select

iif(charindex(' ',trim(@name))>0,Substring(trim(@name),1,charindex(' ',trim(@name))-1),trim(@name)) as First\_Name,

iif(@spaces<>2,'No Middle Name',substring(trim(@name),charindex(' ',trim(@name)),

charindex(' ',reverse(trim(@name)))+1)),

iif(charindex(' ',trim(@name))>0,reverse(substring(reverse(trim(@name)),1,charindex(' ',reverse(trim(@name)))-1)),'No Last Name') as last\_Name

76)Replace String

we use replace on string it does changes the existing string it just display string by applying changes on old string. same like python.

in above example we replace spaces but @name variable does not changes. it shows strings with changes we applied on it.

77)Index of a character a specified occurance

Declare @name as varchar(100)='Pavan kumar'

declare @index as int=1

declare @occurance as int=3

declare @search as varchar(100)='a'

declare @count as int=0

declare @val as int=len(@name)

while @index<=@val

begin

    declare @char as char(1)=substring(@name,@index,1)

    if @char=@search

    begin

        set @count=@count+1

    End

    if @count=@occurance

    Begin

        select @index

   End

    set @index=@index+1

End

78)Use of Over Clause

Over clause used to define the window for the window functions or aggregate functions it’s main aim is to define window for the function. In over clause we can specify the window by using partition by order by and using frame.

Window means particular set of rows in result.

79)Same Value for partition

select Id,Emp\_Name,Gender,Salary,Dept\_ID,Location\_Id,

Dense\_rank() over (order by gender)

as Part from employee

we get same value for whole partition like all male get 2 and all female get 1.

80)IF, Else If, Else

Declare @Marks as int=80

if @marks>=90

Begin

    Select 'A+' as grade

End

Else If @Marks>=80

Begin

    Select 'A' as grade

End

Else If @Marks>=80

Begin

    Select 'A' as grade

End

Else If @Marks>=70

Begin

    Select 'B+' as grade

End

Else If @Marks>=60

Begin

    Select 'B' as grade

End

Else If @Marks>=50

Begin

    Select 'C+' as grade

End

Else If @Marks>=40

Begin

    Select 'C' as grade

End

Else

Begin

    Select 'Fail' as grade

End

81)Nested IF Else.

Declare @pin as int=4267

Declare @pin\_entered as int=426

Declare @amount as int=30000

Declare @amount\_need as int=2000

Declare @ac\_no as varchar(100)='asdfghjkl123'

Declare @ac\_no\_entered as varchar(100)='asdfghjkl123'

if @ac\_No=@ac\_no\_entered

Begin

    if @pin=@pin\_entered

    Begin

        if @amount>@amount\_need

        Begin

            Select 'Trnasaction Completed'

        End

        Else

        Begin

            Select 'Insufficient Funds'

        End

    End

    Else

    Begin

        Select 'Pin\_Incorrect'

        Declare @pinchange as varchar(100)='Y'

        if @pinchange='Y'

        Begin

          if @amount>@amount\_need

          Begin

              Select 'Trnasaction Completed'

          End

          Else

          Begin

              Select 'Insufficient Funds'

          End

       End

     End

  End

 Else

 Begin

    select 'AC no is invalid'

 End

82)While loop

Declare @val as int=1

While @val<=100

Begin

    Select @val as [value]

    set @val=@val+1

End

83)Why Truncate is DDL

When we truncate data from the table a session level commit happened on it so we cannot rollback the truncate and delete can be rolled back and when we truncate the data it resets the identity to its starting value which is a part of object definition

84)Filtering in Where Clause

Always filter data in where clause rather than having clause because let take a scenario we need to group the data based on dept\_id for dept\_id in 1,2,3,4,5 where we have 10 departments we can achieve this by two ways

Select Dept\_Id,Sum(Salary) as Sal From employee

Group by Dept\_Id Having Dept\_Id in (1,2,3,4,5)

Select Dept\_Id,Sum(Salary) as Sal From employee

Where Dept\_Id in (1,2,3,4,5)

Group by Dept\_Id

In first case after grouping the data filtering is happening so SQL engine first need to do aggregate data the it apply filter it takes more time, In second case before aggregated we filtering data so takes less time.

85)Upsert

create table Emp (id int,Name varchar(100),Salary int,Gender varchar(10))

insert into emp values (1,'Pavan',30000,'M'),(2,'Sasi',32000,'M'),(3,'Babavali',33000,'F')

Declare @id as int=5

Declare @Name as varchar(100)='naveen'

Declare @Salary as int=40000

Declare @gender as varchar(100)='F'

Merge Emp T

Using (values(@id,@name,@salary,@gender)) as S(id,Name,Salary,Gender)

on T.id=S.id and t.name=s.name and t.gender=s.gender

When Matched Then

update set t.id=s.id,t.name=s.name,t.Salary=s.salary,t.Gender=s.gender

When not matched then

insert (id,name,salary,gender) values(s.id,s.name,s.salary,s.gender);

select \* from emp

86)Update one table with other table/using from clause and join in update

select \* from Persons

select \* into persons from persons2

select \* from persons2

Select \* From AddressList

update persons

set persons.personpostcode=addr.postcode,

persons.personcityname=addr.city

from persons per

inner join

AddressList addr

on per.PersonId=addr.PersonId

merge persons2 p

using addressList a

on p.PersonId=a.PersonId

when matched then

update set p.PersonPostCode=a.PostCode,p.PersonCityName=a.City;

Merge statement must end with semi colon(;)

87)Try\_Cast, Try\_Convert, Try\_Pharse

Cast, Covert return error if conversion failed where Try\_Cast,Try\_Convert,Try\_Pharse return null if we it unable to convert.

Try\_Pharse use to convert string data to numeric and date (XML no possible)

Try\_Cast and Try\_Covert same as Cast and Convert.

Select Try\_Cast('10' as int)

Select Try\_Cast('pavan' as int)

Select Try\_Convert(int,'10')

Select Try\_Convert(int,'pavan')

Select Try\_Convert(date,'2022-02-02',102)

Select Try\_Parse('2022-02-02' as date)

select Try\_Parse('10.23' as float)

Select Try\_parse('<root></root>' as xml) --error

88)Exists vs In/Not Exists vs Not In

|  |  |
| --- | --- |
| Exists/Not Exists | In/Not In |
| Exists gives true if any value is returned by subquery.  Not exists return true if no value return by sub query | In used as replace of multiple or  Not in used as replace of multiple and |
| We use Correlated Sub Query | Non-Correlated Subquery |
| Slow Comparatively | Faster |
| Not Exists works fine with nulls | Not Null returns empty result if there is null |
| We cannot use hard core values in Exits/Not Exists | We can hard core values. |
| We can do multiple columns match in one Exists | We can do one column match in one in. |

Select \* From Employee E

where exists(Select dept\_id From department D Where E.dept\_id=D.dept\_id)

Select \* From Employee

Where Dept\_Id in (Select Dept\_Id From department)

Select \* From Employee Where Dept\_Id in (1,2,3)

Select \* From Employee where Dept\_ID not in (1,2,3,null)

 Select \* From Employee E

where exists(Select dept\_id From department D Where E.dept\_id=D.dept\_id and E.sub\_Dept\_ID=D.Sub\_Dept\_Id)

Select \* From Employee

Where Dept\_Id in (Select Dept\_Id From department)

and sub\_Dept\_Id in (Select Sub\_Dept\_Id From Department)

Note: -In Exists and in we get records from one table only where in inner join we get records from both tables

89)Magical Tables outside trigger

Magical Tables means inserted and deleted mostly we use them in trigger.

Inserted stored inserted records if new records are inserted then it does not hold previous records.

Same as deleted it holds deleted records when updated, updated record moved to inserted and old records moves to deleted if any new update occurs then old records will disappear.

Inserted

insert into table1 (id,name,gender)

output inserted.id,inserted.name,inserted.gender

into table2

values(1,'pavan','m')

insert into table1 (id,name,gender)

output inserted.id,inserted.name,inserted.gender

into table2

values(2,'sasi','m'),(3,'babavali','m')

identity value also stored in inserted

create table a(id int identity,name varchar(100),gender varchar(100))

create table b(id int ,name varchar(100),gender varchar(100))

insert into a (name,gender)

output inserted.id,inserted.name,inserted.gender

into table2

values ('pavan','m')

Deleted

Delete From table1

output deleted.id,deleted.name,deleted.gender

into table2

where id=1

Delete From table1

output deleted.id,deleted.name,deleted.gender

into table2

where id in(2,3)

insert into table1 values(1,'pavan','m')

Update (inserted/Deleted)

update table1 set name='murali'

output cast(inserted.id as varchar(100))+inserted.name+inserted.gender,

cast(deleted.id as varchar(100))+deleted.name+deleted.gender

into table3

where id=1

update table1 set name='murali'

output cast(inserted.id as varchar(100))+inserted.name+inserted.gender,

cast(deleted.id as varchar(100))+deleted.name+deleted.gender

into table3

where id in (1,2,3)

create table table3(New\_record varchar(100),Old\_record varchar(100))

Select \* From table3

90)Window Functions- Window Functions are the functions which performs

operation based on the particular rows of result set.(defined window)

over- over clause used to define the windows for the window functions and aggregate function by using

partition by, order by and row/Range.

row\_number-row\_number gives unique number to the each row in the defined window.

Select \*,

row\_number() over (Partition by EmployeeID order by orderID) as Row\_numbers

From Orders

Rank - rank is the window function which gives the rank to the each records based on value in a column on defined widow.

Select \*,

rank() over (partition by employeeID order by (select 1)) as [rank]

from orders --every record get rank=1

Select \*,

rank() over (partition by employeeid order by freight desc) as [rank]

from orders

Dense\_Rank-Dense\_rank is the wndow function which gives the rank to the each records based on value in a column on defined widow

Dense\_rank is same as rank but it not sikps the ranks

eg:- if rank gives 1,1,3,4,4 where dense\_rank gives 1,1,2,3,3

where row\_number gives as 1,2,3,4,5

Lag - lag is a window function which get the previous specifed records

with cte

as

(

select year(shippedDate) as [year],month(ShippedDate) as [month],sum(Freight) as Freight

from orders

group by year(shippedDate),month(ShippedDate)

)

Select \*,

lag(Freight,12,0) over (order by [year],[month]) as prev

From cte

where [year] is not null

--lag(column,values\_position,default)

--value position default 1

Lead- lead is a window function which return the next specified records.

with cte

as

(

select year(shippedDate) as [year],month(ShippedDate) as [month],sum(Freight) as Freight

from orders

group by year(shippedDate),month(ShippedDate)

)

Select \*,

lead(Freight,12,0) over (order by [year],[month]) as prev

From cte

where [year] is not null

--lead(column,values\_position,default)

--value position default 1

Ntile- it is window function which used to split the results in to n number of parts

Select \*,

Ntile(8) Over (Partition by EmployeeId Order by employeeID) as [Ntile]

From Orders

eg 10 rows 3 parts 3,3,4

Select \*,

Ntile(8) Over (Order by employeeID) as [Ntile]

From Orders

first\_value is a window function which return the first records in the defined window.

Select \* ,

first\_value(CustomerID) over (partition by year(orderDate),month(orderDate)

order by year(orderDate)) as FirstValue

From Orders

Last\_value is a window function which return the last value in the defined window.

Select \* ,

first\_value(CustomerID) over (partition by year(orderDate),month(orderDate)

order by year(orderDate)) as FirstValue

From Orders

91)Frame Clause

Select \* ,

first\_value(CustomerID) over (partition by year(orderDate),month(orderDate)

order by year(orderDate) rows between 3 preceding and  current row

) as FirstValue

From Orders

Select \* ,

last\_value(CustomerID) over (partition by year(orderDate),month(orderDate)

order by year(orderDate) rows between current row  and  3 following

) as lastValue

From Orders

Select \* ,

first\_value(CustomerID) over (partition by year(orderDate),month(orderDate)

order by year(orderDate) rows between 3 preceding and  current row

) as FirstValue ,

last\_value(CustomerID) over (partition by year(orderDate),month(orderDate)

order by year(orderDate)  rows between current row  and  3 following

) as lastValue

From Orders

Create table ranges( id int identity,name varchar(100),salary int)

insert into ranges values

('pavan',20000),('sasi',28000),('babavali',30000),('babavali',31000)

select \*,

last\_value(salary) over (order by name desc) fv2,-- range between unbounded preceding and current row) as fv1,

last\_value(salary) over (order by name desc rows between unbounded preceding and current row) as fv2

from ranges

Frame clause -In Frame clause we use row and range in over clause to define the window(particular rows in result set)

on which function operate.

by default frame is range between unbounded preceding and current row

preceding -previous rows

following - next rows

current row

range allows only unbounded preceding and current following which is default.

row is nothing but a record n result.

main difference between range and row is when we have duplicate range consider duplicates as single unit

where row does not consider duplcates as single unit it treats as individual rows

row allows preceding and following where range allows only unbounded preceding and current following which is default.

92)Instead of Triggers

This trigger before the action performed on table.

There are insert, update and delete.

Create table Base\_Table

(

  id int,

  name varchar(100),

  gender varchar(100),

  age int,

  salary INT

)

Alter Trigger [Instead\_of\_insert]

on Base\_Table

INSTEAD of insert

AS

Begin

    Declare @id int, @salary int,@age int,@name varchar(100),@gender varchar(100)

    select @id=id,@name=name,@age=age,@gender=gender,@salary=salary From Inserted

    if len(@name)>=5

    Begin

        if @gender in ('M','F')

        Begin

            IF @Age>18 AND @Age<100

            Begin

                insert into base\_table values (@id,@name,@gender,@age,@salary)

            End

        End

        Else

        Begin

            Select 'Enter Correct gender'

        End

     End

     Else

     Begin

        Select 'Enter name having length equal and greater then 5 charcters'

     End

End

insert into Base\_table values (1,'Pavan','M',24,40000)

select \* from Base\_Table

Create Trigger [Instead\_of\_Delete]

on Base\_table

INSTEAD of Delete

as

Begin

    Declare @id as int, @name as varchar(100),@gender as varchar(100),@age as int,@salary as int

    Select @id=id From Deleted

    if @id=1

    Begin

        Select 'This Record Cannot be deleted'

    End

    Else

    Begin

        Insert into Base\_table values (@id,@name,@gender,@age,@salary)

    End

End

Delete From base\_table where id=1

insert into Base\_table values (2,'Pavan','M',24,40000)

Delete From base\_table where id=2

Alter Trigger [instead\_of\_update]

on Base\_Table

instead of update

as

Begin

    Declare @id1 as int,@id2 as int

    select @id1=id from inserted

    select @id2=id from deleted

    if @id1<>@id2

    Begin

        select 'Does not allow Update'

    End

    Else

    Begin

        Select 'You can Update'

    End

End

Update Base\_Table set id=2 where id=1

Update Base\_Table set name='sasi' where name='pavan'

93)How Cluster Index stored and works

Cluster index stored as B+ tree structure.

It consist of root, intermediate nodes, and leaf nodes.

Let consider employee table has 1000 employees and we created clustered index on id then they stored as following.

First root consist of 1 to 1000 next intermediate rows consist of 1 to 250, 251 to 500, 501 to 750, 751 to 1000

And in next intermediate 1 to 125, 126 to 250, 251 to 375, 376 to 500, 501 to 625, 626 to 750, 751 to 875 and 876 to 1000 as so on.

When we need find 623 record first it goes to 501 to 1000 form root node then 500 to 750 the 500 to 625 then it find 623 record and return that record. Because in leaf node we have actual data.

94)How Non-Clustered Index Works and Stores

Non-Clustered index stored as B+ tree.

It consist of key values in leaf node we have row locaters that consist of key value (on which non clustered index created) as a clustered key mostly primary key (clustered index column).

Let consider employee table have 1000 records and have 1000 employee names along with employee name their id also stored in key locater.

We want pavan record whose id is 455 then it find pavan in non-cluster index then we have pavan and 455 SQL engine use cluster in index on id to locate that record and uses inner join to retrieve all data of employee whose name is pavan.

95)Execution Plan

Execution plan is the plan which is used or followed by SQL Engine to execute query.

Read execution plan right to left and top to bottom.

By using execution plan we can analysis the performance of query and we can identity bottlenecks for trouble shooting.

There are two Execution plans Actual Execution plan and Estimated Execution plan.

Actual Execution plan gives plan which is followed by SQL Engine where estimated Execution plan gives plan which may be followed by SQL Engine. By using estimate Execution plan we can find bottlenecks with out executing.

Datetype of column.

type\_Name return the datatype name with it user\_type\_id. we get user\_type\_id from sys.columns.

select \*,TYPE\_NAME(user\_type\_id) as type from sys.columns where [object\_Id]=object\_Id('Employee')

Cascade referential integrity constraint.

We can define the type of action to perform when ever an related column is deleting or getting update.

i.e. when a row in department table is deleting which is related to the

dept\_id in employee we can set the action to be performed by cascade referential intigrity constraint.

default it resets the related values in employee table with default value. if default is not specified it replaces null.

null it set related values in employee table as null

no action it does not allow to remove row from dept table throws error.

cascade it deletes related records from employee table.

Alter Sequence

Create Sequence dbo.SQE1 as int

start with 1

increment by 1

minvalue 1

maxvalue 100

cycle

cache 4

select next value for dbo.SQE1

Alter Sequence dbo.SQE1

Restart with 1

increment by 1

minvalue 1

maxvalue 100

cycle

cache 3

Dateadd and datediff result

select datediff(day,'2024-01-01','2024-01-30')

select datediff(month,'2024-01-01','2024-01-30')

in datediff we get 30-01=29 not 30 same for month and year

select dateadd(day,3,'2024-01-01')

select dateadd(day,-3,'2024-01-01')

in dateadd we need to add 3 days for 01 we get 04 it adds 02,03,04 not considers 01.

Create Clustered index CI on Employee(ID asc)

Create NonClustered index NCI on Employee(ID asc)

we can create clustered and non clustered index on same column.

Create NonClustered index NCI1 on Department(Dept\_id asc)

Create Clustered index CI1 on Department(Dept\_id asc)

we can create non clustered index before creating clustered index.

If table on which indexes are created is deleted then index also deleted.

Create Table q(id int)

Create clustered index ci2 on q(id asc)

drop table q

Create Table q( id int,name varchar(100),Gender varchar(100),cgpa float,salary int)

alter table q add constraint uk1 unique(id,name)

alter table q add constraint df1 default 10 for cgpa,

constraint df2 default 100 for salary

so we can add multiple constraint at same time by using comma.

alter table q

add constraint df3 default 'M' for Gender,

new int null

we can add constraint and column in one statement

we cannot alter multiple columns at a time.

alter table Employee add constraint fk foreign key(dept\_id) references department(dept\_id)

on update set null on delete set default

foreign key with referential integrity cascade constraint

we cannot alter constraint by sql command.

to alter we need to delete constraint and add constraint with same name with new changes.

Recursive Stored Procedure

Create Proc Rec\_Stored(@Dept\_ID as int)

as

Begin

if @Dept\_ID<10

Begin

Select \* From Employee Where Dept\_Id=@Dept\_Id

Set @Dept\_Id=@Dept\_Id+1

Exec Rec\_Stored @Dept\_Id

End

End

Exec Rec\_Stored 4

here stored procedure refers it selfs untill conditions satisfies.

Difference Between Char, Varchar, Nvarchar,Varchar(Max)

Char stores fixed length if we declare variable with char(10) if we use 5 bytes then also it takes 10 bytes of memeory.

varchar it is variable length if we declare a variable with varchar(10) we use  bytes it takes  bytes of memeory only.

NVarchar it stores unicode data as well where varchar store asci values. but Nvarchar take 2\*Bytes

Varchar(Max) we use this when we donot know much data we going to insert to that column or variable. it takes data up to (2^31)-1

it recommended to use if data exceeds 8000 bytes.

isnumeric

isnumeric return 1 if values in numeric or convertable to numeric

and return 0 if it is not numeric.

Select ISNUMERIC('10')

Drop statements

Drop table [Table Name]

drop index [Table Name].[Index Name]

drop trigger [Name of Trigger] --DML Trigger

drop trigger [Name of Trigger] on [scope All Server/Database] --DDL Trigger

drop view [Name of View]

drop

Deallocate [Name of Cursor]

Drop Function [Name Of the Function]

Drop Proc [Name of the Proc]

Alter Table Q Drop Constraint uk1

listing objects in databse

Select \* From Sys.Objects where type\_desc like '%Proc%' --procedure

Select \* From Sys.Objects where type\_desc like '%user\_Table%' --table

Select \* From Sys.objects where type\_desc like '%Func%' --function

Select \* From Sys.Objects where type\_desc like '%Func%' and type\_desc like '%scalar%' --scalar function

Select \* From Sys.Objects where type\_desc like '%Func%' and type\_desc like '%inline%' --inline table valued function

Select \* From Sys.Objects where type\_desc like '%Func%' and type\_desc like '%multi%' --multi statment table valued function

Select \* From Sys.Objects where type\_desc like '%View%' --view

Select \* From Sys.Objects where type\_desc like '%trigger%'  --trigger

Select \* From Sys.Objects where type\_desc like '%constraint%' and type\_desc like '%Check%' --check constraint

Select \* From Sys.Objects where type\_desc like '%constraint%' and type\_desc like '%Unique%' --unique constraint

Select \* From Sys.Objects where type\_desc like '%constraint%' and type\_desc like '%PRIMARY%' --primary key constraint

Select \* From Sys.Objects where type\_desc like '%constraint%' and type\_desc like '%Foreign%' --foreign key constraint

Select \* From Sys.Objects where type\_desc like '%constraint%' and type\_desc like '%default%' --default constraint

Select \* From Sys.Objects where type\_desc like '%constraint%' --constraint

Select vs Print

Select is SQL command which used to retrive data from database objects.

where print is transact sql command which used to print the message(length up to 8000 for non unicode and 4000 for unicode) or values in a variable.

Select return result in results tab where print return result in message tab.

Find Masked Columns

Alter Table Employee Alter column Salary  add masked with (function='default()')

Alter Table Employee Alter column id int masked with (function='random(1,100)')

Select \* from sys.masked\_columns where [object\_id]=object\_id('Employee')

Difference between is and equal

IS used to campare with nulls where = used to compare with values.

Note:-Byte is a unit of memory which consist on 8 bits. bit is a dataype it stores 0 or 1 boolean.

Foreign key refer multiple primary keys.

create table empnew1(id int primary key,Name varchar(100),Location\_id int,Dept\_id int,gender varchar(100),

constraint fkdept1 foreign key(dept\_id) references dept1(id),

constraint fkdept2 foreign key(dept\_id) references dept2(id))

Create table dept1( id int primary key, Name Varchar(100),Head\_Id int)

Create table dept2( id int primary key, Name Varchar(100),Head\_Id int)

insert into dept1 select Dept\_Id,Dept\_Name,Dept\_Head\_Id From DEPARTMENT where dept\_id<7

insert into dept2 select Dept\_Id,Dept\_Name,Dept\_Head\_Id From DEPARTMENT where dept\_id>=7

insert into empnew1

select id,Emp\_Name,Location\_Id,Dept\_Id,Gender From Employee where dept\_id=8

insert into empnew1

select id,Emp\_Name,Location\_Id,Dept\_Id,Gender From Employee where dept\_id=1

we can create two foreign key on dept\_id one with dept1 and one with dept2 but when we insert data it throw error if we enter dept\_id in dept1

then dept2 fk throws error if we enter dept id in dept2 then dept1 fk throw error.

create table empnew2(id int primary key,Name varchar(100),Location\_id int,Dept\_id int,gender varchar(100))

insert into empnew2

select id,Emp\_Name,Location\_Id,Dept\_Id,Gender From Employee

Select \* From empnew2

to overcome this create new table and insert both table records into new table and create a flag to distingush records between two records in table.

Select \* From dept1

Select \* From dept2

Select e.id,e.Name,Location\_id,Dept\_id,

gender,isnull(d.name,d2.name) as [Dept\_Name],

isnull(d.Head\_id,d2.Head\_id) as Head\_Id

From empnew2 e

left join dept1 d on e.Dept\_id=d.id

left join dept2 d2 on e.Dept\_id=d2.id

Cast vs Convert

Cast and Convert used to change datatype which are compatible means 'pavan' cannot be convert into int.

we can specify style in Convert which helps to get data in different formats. (102,101 etc).

Cast is ANSI SQL where Convert is T-SQL.

Select cast('2022-01-01' as date)

Select cast('99' as int)

Select Convert(date,'2022-01-01',102)

Select Convert(int,'22')

SQL Injections

Declare @Table\_Name as varchar(100)='Employee; drop table ED --'

Declare @SQL as varchar(max)='Select \* From '+@Table\_Name

Select @SQL

Execute (@SQL)

to avoid this we use quotename

go

Declare @Table\_Name as varchar(100)='Employee; drop table DUP --'

Declare @SQL as varchar(max)='Select \* From '+quotename(@Table\_Name)

Select @SQL

Execute (@SQL)

Fill Down

with cte

as

(

select \*,count(Salary) over (order by id) as cou From fud

)

Select id,name,gender,salary,

isnull(first\_value(Salary) over (partition by cou order by id),0) as fill\_down from cte

referencing and referenced entities

Select \* From Sys.Dm\_SQL\_referencing\_entities ('dbo.Employee','Object') --to get all entities which are refering specified object

Select \* From Sys.Dm\_SQL\_referenced\_entities ('dbo.Return\_Sum\_Salary','Object') --to get all entities with columns on which it depended

if we create a procedure on a table procedure is referencing entity and table is referenced entity. here procedure refering table.

so table is refered and procedure is referencing.

Geometry

Geometry is a datatype which used to represent shapes on the 2d plane.

shapes like point,circle,line etc.

spatial referel id which used to define the shape of earth. like sperical, flat earth etc.

it is zero for geometry because it represents on 2d plane.

create table geom

(

id int identity,

name varchar(100),

xy geometry

)

select \* from geom

insert into geom values ('point',Geometry::STGeomFromText('point(2 3)',0))

insert into geom values ('MultiPoint',Geometry::STGeomFromText('multipoint(2 3,4 5)',0))

insert into geom values ('Line String',Geometry::STGeomFromText('LineString(2 3,4 5)',0))

insert into geom values ('MultiLineString',Geometry::STGeomFromText('MultiLineString((3 4,5 6),(5 6, 4 4),(4 4,3 4))',0))

insert into geom values ('circularstring',Geometry::STGeomFromText('circularstring(2 2,2 -2,-2 -2,-2 2,2 2)',0))

insert into geom values ('polygon',Geometry::STGeomFromText('polygon((2 2,-2 2,-2 -2,2 -2,2 2))',0))

insert into geom values ('polygon2',Geometry::STGeomFromText('polygon((3 3,-3 3,-3 -3,3 -3,3 3),(1 1,-1 1,-1 -1,1 -1,1 1))',0))

Select xy.STGeometryType() From Geom

Select xy.STStartPoint().ToString() From Geom

Select xy.STEndPoint().ToString() From Geom

Select xy.STPointN(3).ToString() From Geom

Select xy.STNumPoints() From Geom

Select xy.STLength() From Geom

Select xy.STBoundary().ToString() From Geom

Declare @xy1 as geometry,@xy2 as geometry

select @xy1=xy from Geom where id=6

Select @xy2=xy from Geom Where id=7

select @xy1.STUnion(@xy2).ToString()

select @xy1.STIntersection(@xy2).ToString()

select @xy1.STDistance(@xy2)

ToString() used to make points readable. it used for functions which need to display coordinates (points)

Geography

geography is also a datatype

it used to represent spaes by considering the curvature of earth. i.e 3d plane

instead on x and y we mention lattitude and longitude.

here in spatial referal id 4236 mostly used (current shape of earth)