```
--DBA THEORIES
--1)CONNECTION PROBLEM
/*red mark in the connection symbol and then the it is not able to connect
then goto start--services.msc--sql server service --(you will find service is stopped)--start it*/


SELECT *
FROM [dbo].[Account_Details]

SELECT *
FROM [dbo].[Transaction_Details]

--basic retrieval from the table

SELECT a.Customer_First_name+'   '+a.Customer_last_name as name, t.Transaction_Amount
from Account_Details as a
inner join
Transaction_Details as t
on a.Customer_ID=t.Customer_ID

--highest value retrieval



select max(Transaction_Amount) as max_amount
from Transaction_Details

--3 highest amounts

select TOP 3 a.Customer_First_name, a.Customer_last_name, t.Transaction_Amount
from Transaction_Details as t
inner join Account_Details as a
on t.Customer_ID=a.Customer_ID
order by t.Transaction_Amount desc

DROP VIEW THIRD_HIGHEST_AMOUNT

--4 3 rd highest amount
Create view THIRD_HIGHEST_AMOUNT
as
select a.Customer_First_name, a.Customer_last_name, a.C_Address, a.Contact_no, t.Transaction_Amount, t.     ↙
    Transaction_Date
from Transaction_Details as t
inner join Account_Details as a
on t.Customer_ID=a.Customer_ID
where Transaction_Amount=(select min(Transaction_Amount)
     from Transaction_Details
     where Transaction_Amount in ( select TOP 3 Transaction_Amount
                                   from Transaction_Details
                                   order by Transaction_Amount desc)
)

select *
from THIRD_HIGHEST_AMOUNT


--5 no of transactions of each customer
         select Customer_ID, count(*) as no_of_transactions
         from Transaction_Details
         group by Customer_ID

--6 date difference and date part


select DATEDIFF(mm,Transaction_Date,GETDATE())
```

```
FROM Transaction_Details
where Customer_ID=1



--7 date part
select datepart(mm,Transaction_Date) as int
from Transaction_Details

select datepart(dd,Transaction_Date) as int
from Transaction_Details

--8 between

select *
from Transaction_Details
where Transaction_Amount between 1000 and 2000

select *
from Transaction_Details
where DATEPART(mm,Transaction_Date) between 05 and 12


--9 case

select *, customer_type =
case
when Transaction_Amount>=2000 then 'GOLD'
WHEN Transaction_Amount<2000 and Transaction_Amount>1000 then 'silver'
when Transaction_Amount<1000 then 'bronze'
end
from Transaction_Details


--10
select *
from Transaction_Details
where Transaction_Type =(select Transaction_Type
                        from Transaction_Details

                        where Transaction_Amount=(Select max(Transaction_Amount)
                                                  from Transaction_Details)
                        )
--11 derieved table

SELECT *
FROM [dbo].[Account_Details]

SELECT *
FROM [dbo].[Transaction_Details]

select top 2 Transaction_Amount, Transaction_Date, Transaction_Type
from (select *
      from Transaction_Details
      where Transaction_Amount>1000) as new

select datepart(mm,Transaction_Date) as highest_transaction_month, Customer_ID
from (select *
      from Transaction_Details
      where Transaction_Amount=(select max(Transaction_Amount)
                                from Transaction_Details) ) as date
    --display name and no of transactions
select  a.Customer_ID,a.Customer_First_name+'     '+a.Customer_last_name, no_of_transactions
from Account_Details as a
join (select Customer_ID, count(*) as no_of_transactions
      from Transaction_Details
```

```
      group by Customer_ID) as b
      on a.Customer_ID=b.Customer_ID
      /*this is prefered table*/

    /*who has transactions
    select*from Account_Details as a
    join Transaction_Details as t
    where exists(select count())*/


    --ranking functions

/*row number, rank*/

select A.Customer_ID, T.Transaction_Amount, ROW_NUMBER() over (order by T.Transaction_Amount desc) as
    row_no
from A_Details AS A
JOIN Transaction_Details AS T
on A.Customer_id=T.Customer_ID


--RETRIEVE THE 5 TH ROW-- USE DERIEVED TABLE(IT IS FROM FROM CLAUSE)
select Customer_ID
from
           (select A.Customer_ID, T.Transaction_Amount,
                  ROW_NUMBER() over (order by T.Transaction_Amount desc) as row_no
                  from A_Details AS A
                  JOIN Transaction_Details AS T
                  on A.Customer_id=T.Customer_ID) as new

where row_no=5

--
SELECT Transaction_Amount, Transaction_Date, Customer_ID, ROW_NUMBER() over(partition by Transaction_Type
    order by Transaction_Amount desc)
from Transaction_Details




use firstprog
go
-- all combinations /*used in ssas*/

SELECT *
FROM items

select item, color, sum(number)
from items
group by item, color

/*does not retrieve all combinations*/

select item, color,sum(number) as qty
from items
group by item, color with cube

/*--different combinations possibles --huge datait is mainly helpful in calculations--olap data*/

select item, color, sum(number) as qty
from items
group by item, color with rollup

/*--subset of cube--no permutation combinations*/
```

```
select item, sum(number)
from items
group by item

select color, sum(number)
from items
group by color

/*only no of items and colors*/

SELECT *
FROM dbo.ENGG_STUDENTS


select name, roll_no, ROW_NUMBER() over(order by marks) as rankers
from dbo.ENGG_STUDENTS

select name, roll_no, branch, ROW_NUMBER() over(partition by branch order by marks) as rankers
from dbo.ENGG_STUDENTS


select *
from(
          select name, roll_no, ROW_NUMBER() over (partition by branch order by marks desc)as ranks
          from dbo.ENGG_STUDENTS) as g
where ranks=1

/*first ranker from each branch*/


select name, roll_no, RANK() over (order by marks desc)as rank_order, DENSE_RANK() over(order by marks
    desc) as rank_dense_order
from dbo.ENGG_STUDENTS
--SO USE DENSE ORDER RATHER THAN RANK

--HIGHEST MARKS

SELECT *
FROM
(select name, roll_no, RANK() over (order by marks desc)as rank_order, DENSE_RANK() over(order by marks
    desc) as rank_dense_order
from dbo.ENGG_STUDENTS)AS RANKER
WHERE rank_dense_order=5

--Branch wise ranking

select *
from(
          select name, roll_no, RANK() over (partition by branch order by marks desc)as ranks
          from dbo.ENGG_STUDENTS) as g
where ranks=1



 select name, roll_no, branch, RANK() over (partition by branch order by marks desc)as ranks, ntile(2) over
     (partition by branch order by marks desc) as group_no
          from dbo.ENGG_STUDENTS

select name, roll_no, branch, ntile(2) over (order by marks asc)as group_no
          from dbo.ENGG_STUDENTS
/*

declare @a int --declare the variable
set @a=20--stores value of the variable
set @a=-23--now 20 will be deleted--only single value
```

```
select @a--will give the value in table
print @--will return the value

declare @name varchar(100)
set @name='harish'
set @name='rahul'--now harish will be deleted--only single value


declare @variable1 int
set @variable1=203

declare @variable2 varchar

select @variable2=name
from table_name
where id=@variable1

print @variable2

*/

declare @x varchar(100)
set @x='harish'

select @x

declare @y int
set @y=349
set @y=@y+49

print @y
select @y as number


declare @name varchar(100)
set @name='harish'
set @name=@name +' kumar'

print @name

declare @z int
set @z=39
set @z=@z-29

select @z

declare @apple int
set @apple=30

print @apple

declare @dob datetime
set @dob='2009/08/07'

print @dob

--this is local variable--available only for this window

use firstprog
go
declare @type varchar(50)
SET @type='SB'

select *
from dbo.Account_Details
WHERE Account_type=@type
```

```
--using variables in queries

declare @acc_type varchar(50), @amount int
set @acc_type='SB'
set @amount=2000

select *
from Account_Details
where Account_type=@acc_type and Amount=@amount

declare @cid int
set @cid=2
declare @amounts int
/*
select @amounts=Amount
from Account_Details
WHERE Customer_ID=@cid

PRINT @amounts*/

declare @name varchar(100)

select @amounts=Amount, @name=Customer_First_name+' '+Customer_last_name
from Account_Details
where Customer_ID=@cid

print @amounts
print @name

select @amounts as amount, @name as name

--2 variables --while printing 2 variables are not taken-- but it does in select statement

select @@VERSION as version_name

--to know the version name

select @@SERVERNAME as servername

--to know the server name

--**imp

select @@IDENTITY as identityValue
from customer_details

select @@ERROR as error_report
from customer_details

select * from
customer_details
go
select @@ERROR as error_report
go

select *from A_Details
GO
update A_Details
set C_Address='DUBAI'
WHERE C_Address='UAE'
GO
select @@ROWCOUNT

select @@CONNECTIONS
select @@MICROSOFTVERSION
```

```
--what is difference between local and global--@ symbol CREATE VARIABLE @@ symbol NO NEED TO CREATE VARIABLE

/*IF AND ELSE*/ --SYNTAX
                              /* DECLARE @VARIABLE
                              SET @VARIABLE=1990098978
                               IF<CONDITION>
                               BEGIN
                               SELECT/PRINT
                               END*/
select *
from Transaction_Details

declare @i INT
set @i= 2

IF((select count(*) from Transaction_Details where Transaction_Type='CW')>@i)
begin
print 'greater'
end
else
begin
select count(*) from Transaction_Details where Transaction_Type='CW'
end


declare @a int
set @a=5
declare @b int
set @b=6
if(@a>@b)
begin
print @a
end
else
print @b


declare @custID INT
set @custID=3

if exists (select * from Transaction_Details where Customer_ID=@custID)
begin
select * from Transaction_Details where Customer_ID=@custID
end
else
begin
select Customer_ID, COUNT(*) FROM Transaction_Details group by Customer_ID
print 'no transaction '
INSERT INTO  Transaction_Details(Transaction_ID,Transaction_Amount,Transaction_Type,Transaction_Date, Customer_ID)
VALUES (7, 3000, 'CL', '08-06-2017', @custID);
select @@ROWCOUNT as row_change
select * from Transaction_Details
end


--while loop---for loop is not compatible with the sql

/* loop requirements
   declare variable
   intialize variable
   condition
   statement
   inc/dec
```

```
*/

--print all number n

declare @m int--declare
set @m=0--intiallize

while(@m<=7)--condition
begin
print @m--statement

set @m=@m+1--increment
end
print 'these are top 7 numbers'


--concatinate

----''+@variable

declare @name varchar(50)

set @name='harish'

print @name
select 'sriramula '+@name as name


--cast() and convert()
declare @rollno int
set @rollno =270

--print 'harish '+@rollno--this doesnot work

print 'harish rollno is '+cast(@rollno as varchar(50))
print 'harish rollno is '+convert(varchar(50), @rollno)--same result

declare @date datetime
set @date=getdate()

select convert(varchar(100), @date) as date_time
select convert(varchar(100), @date,1 ) as date_time
select convert(varchar(100), @date, 2)as date_time
select convert(varchar(100), @date, 100) as date_time


select * from [Person].[Address]

--temp tables--just like views but not for temp use --may be local and global--will be tempDB database

create table #person(AddressID INT identity(1,1), AddressLine1 varchar(100), city varchar(100))

insert into #person
values
(1,'marsh ln','texas'),(2,'broadway street','pittsburg'),(3,'wright street','dayton')

 select *from #person

--used by only 1 user and does not be useful out of the window
create table ##person(AddressID INT , AddressLine1 varchar(100), city varchar(100))

insert into ##person
values
('marsh ln','texas'),('broadway street','pittsburg'),('wright street','dayton')

 select *from ##person
```

```
 --can be used by other connections also

 /*views*/

 --used to look daily info --ex: bank manager needs this--he may want to see regional wise data
 --not the entire data --store the query--store any dml statements--view has query but the query cannot be ↙
    used afterwords
 --used for retrieving specific data--security(hackers may go through the database and del db)
 --create views from different db joining--different connection joining
 --can create a view on a view--no temp views
--you cannot use order by statement

create view people
as
select *from [Person].[Person]

select *from people

create view people_midName
as
select *from [Person].[Person]
where MiddleName is not null

select *from people_midName


sp_HELP

--originally people do like tbl_ for tables and v_ for views

--get query

USE [firstprog]
GO
sp_helptext [A_Details]

drop [dbo].[people]


--view cannot store data--the data is stored only by query
--if base table change data, the view data changes
--you can retrieve, create, del data but all these things will be done by query

--types of views securtiy --updatable and non updatable
--if no: of columns are different, can i insert using view? yes, if you the column is nullable and no      ↙
    aggregate
                                                          --no if the column is not nullable and has      ↙
    aggregate


--thick client--big ram/hd/processor
--light client--small ram/hd/processor

--centralised server(db side) --and i have basic server

--once in a month --generate payslips--code will be available in the client side and the data is available ↙
    in db side(centralized side)
--client will generate code(front end) and send to db server --network traffic jam may be possible

--db server ---client 1
            ---client 2
            ---client 3

--ex: produce mini statement
/*
```

```
select *
from tm
where acid=123
order by dot desc
(this is the code written and stored in the atm to get the ministatement)
*/

/*--disadv if the code is near client

network traffic
maintainance of code--code may be changed
security--any body can tamper the code
performance is low--travelling and execution
*/

/*how can i centralize the code--how to keep the code in the server side?
type            input parameter    output/return        ddl    dml    cache execution(unique features like ↙
    index, keys)
view-              no                   no             no     yes       no
sp-                yes                  yes            yes    yes       yes
function-          yes                  yes            no     no        no

faster sp>functions&views
max size 128mb
may have temp sp --local and global

SYNTAX OF STORED PROCEDURE

create procedure procedurename(input parameter, output parameter output)
as
begin
    declare variables
    ddl
    dml
    programming
    call a function
    call another sp
    use cursors
    return statement
    end
go

exec procedurename
*/


--simple proc

create proc welcome
as
begin
          print 'welcome to new sql world'
end
go
exec welcome

select *from [Person].[Person]


create proc getName(@businessID INT)
as
begin
    select FirstName+' '+MiddleName+' '+LastName as Name, PersonType
    from [Person].[Person]
    where BusinessEntityID=@businessID
end
```

```
go
exec getName 1


select *from [Production].[ProductListPriceHistory]
select * from [Sales].[Currency]



/*procedure with 1 input and 1 output and 1 return*/
create proc country(@code varchar(20), @name varchar(50) out)
as
BEGIN
    select @name=Name
    from [Sales].[Currency]
    where CurrencyCode=@code
    RETURN 0
end
go


DECLARE @RC INT
DECLARE @name varchar(20)
EXEC @RC=country 'ALL', @name out
select @RC as returnValue

IF (@RC =0)
BEGIN
    SELECT @name
END

ELSE
BEGIN
    print 'invalid details'
END

/*

syntax of procedure

create procedure procedureName(@parameter datatype in
                               @parameter2 datatype out
                               @parameter3 datatype out)
as
begin

       select @parameter2= columnName2, @parameter3= columnName3
       from tableName
       where columnName=@parameter


end

declare @parameter2 datatype
        @parameter3 datatype
exec procedureName <parameter value>, @parameter2 out, @parameter3
select @parameter2 as column2, @parameter3 as column3


declare out put parameters as variables while calling
drop proc proc_name

see the body/syntax of the procedure

exec helptext <view/function/proc>
```

```
IF YOU WANT TO ALTER THE PROCEDURE THEN SAY ALTER
*/



USE AdventureWorks2012
GO
CREATE PROCEDURE countryName(@CODE nchar(3), @NAME VARCHAR(20) OUTPUT)
AS
BEGIN
SELECT @NAME=Name
from [Sales].[Currency]
where CurrencyCode=@CODE
END
GO
declare @NAME VARCHAR(20)
EXEC countryName 'AED', @NAME out
SELECT @NAME AS NAME

exec sp_helpdb
exec sp_helptext countryName

use firstprog
go
select *from [dbo].[ENGG_STUDENTS]

create procedure students
                          (@class int,
                          @name1 varchar(50) out,
                          @branch varchar(50) out)
as
begin

declare @cnt int
select @cnt= count(*) from ENGG_STUDENTS where class=@class
if (@cnt=0)
begin
    print 'invalid entry'
end
else
begin
    select @name1=name, @branch=branch
    from ENGG_STUDENTS
    where class=@class
end
end

declare @name1 varchar(50)
declare @branch varchar(50)
exec students 1, @name1 out, @branch out
select @name1 as name , @branch as branch

exec sp_helptext students
/*
PARAMETER HAS NAME, TYPE, IN/OUT

FIRST TIME EXECUTION--late

LOOKS IF THERE ARE
joins
keys
indexes
volume
..etc

second>faster than first
```

```
can also create temp proc by #

upto 32 level of calling (nesting)
sp1 calling sp2
sp2 calling sp3
sp3 calling sp4

---------

recursion is also possible--it is also 32 levels

return statement is used to know if the procedure executed properly or not


*/


--recompilation

/*change the execution plan
 if the no:of rows increase, if the query was modified, if the old plan was not able to use no longer
 if the plan changes every time we have to use recompilation*/


USE AdventureWorks2012
GO
alter PROCEDURE countryName(@CODE nchar(3), @NAME VARCHAR(20) OUTPUT, )
AS
BEGIN
SELECT @NAME=Name
from [Sales].[Currency]
where CurrencyCode=@CODE
END
GO
declare @NAME VARCHAR(20)
EXEC countryName 'AED', @NAME out
SELECT @NAME AS NAME

USE AdventureWorks2012
GO
select *from [Sales].[Currency]

--execution plan at creation
USE AdventureWorks2012
GO
alter PROCEDURE countryName(@CODE nchar(3), @NAME VARCHAR(20) OUTPUT )with recompile
AS
BEGIN
SELECT @NAME=Name
from [Sales].[Currency]
where CurrencyCode=@CODE
END
GO
declare @NAME VARCHAR(20)
EXEC countryName 'AED', @NAME out
SELECT @NAME AS NAME

--recompile at execution


USE AdventureWorks2012
GO
alter PROCEDURE countryName(@CODE nchar(3), @NAME VARCHAR(20) OUTPUT )with recompile
AS
BEGIN
```

```
SELECT @NAME=Name
from [Sales].[Currency]
where CurrencyCode=@CODE
END
GO
declare @NAME VARCHAR(20)
EXEC countryName 'AED', @NAME out with recompile
SELECT @NAME AS NAME

exec sp_recompile

/*types of procedures*/
/*
1)system sp

sp_help
sp_helpdb
sp_helptext
sp_indexes
*/
exec sp_help

/*
2)user defined sp

*/

/*
3)extended sp

it can be done out of the sql server
use c/c++
you can create a folder/send a mail

xp_

*/

use master
go
exec xp_logininfo
exec xp_msver
exec xp_CMDShell 'OS Cmd'
exec xp_cmdshell 'MD D:\TEST'
exec xp_SendMail 'hai', 'harish.kumarqa5@gmail.com'


/*CLR stored Procedures (.NET)

--Disadvantage of xp_
* memory leakage
* data cannot be removed from the ram
*c/c++ unmanaged code

use c#.net code for this purpose
*because it is managed code(removes memory when your application is closed)
--to connect to .net
--go to assemblies and register for assembly
*/

/*
nested sp

syntax
```

```
 create proc sp1
 as
 begin
      declare @x int
      exec sp2 103, @x OUT --EXAMPLE
end

create proc sp2
as begin
      exec sp3
end

*/



/*CS--CREATE SP FOR LAON STATEMENT

INPUT:

loan amount
rate of interest
tenure


OUTPUT

sl:no
paydate
emi amount
slno              paydate         emi amount
1                07/08/2017       3839
2.....................................


LOGIC

loan date
formula
interest = pnr/100
total amount =loan amunt+ interest
emi= total amount/tenure(months)


display the above data

*/

/* CS generate payslips using sp */

/* CS generate mini statements */

alter proc LoanStatement
(
@loanAmount money =10000,--default values
@rate int = 14,
@tenureInYears tinyint=2
)
as
begin
--declare a variable for loan date
declare @loanDate datetime
set @loanDate =getdate()

--calculate interest
```

```
--formula =pnr/100

declare @interest money
set @interest =(@loanAmount *@tenureInYears*@rate)/100

--calculate total amount
declare @totalAmount money
set @totalAmount=@loanAmount+@interest

--calculate emi
declare @emi money
set @emi=@totalAmount/(@tenureInYears*12)

--display emi table
print '*************************************************************************'
print 'sl:no'+space(10)+'EMI Date'+space(10)+'EMI Amount in USD'
print '*************************************************************************'

--loop to display all the data

declare @i int
set @i=1

--while condition
while(@i <=(@tenureInYears*12))
begin
        print cast(@i as varchar(20))+
              space(10)+cast(dateadd(mm,@i, @loanDate) as varchar(20))+
              space(10)+cast(@emi as varchar(20))

        set @i=@i+1
end

print '*************************************************************************'
print 'GRAND TOTAL :'+SPACE(10)+cast(@totalAmount as varchar(30))
print '*************************************************************************'
print '*****************THANK YOU FOR VISITING OUR BANK **********************'
end
go



 exec LoanStatement

 /*
*************************************************************************
sl:no          EMI Date          EMI Amount in USD
*************************************************************************
1           Jul 28 2017 11:31PM          533.33
2           Aug 28 2017 11:31PM          533.33
3           Sep 28 2017 11:31PM          533.33
4           Oct 28 2017 11:31PM          533.33
5           Nov 28 2017 11:31PM          533.33
6           Dec 28 2017 11:31PM          533.33
7           Jan 28 2018 11:31PM          533.33
8           Feb 28 2018 11:31PM          533.33
9           Mar 28 2018 11:31PM          533.33
10          Apr 28 2018 11:31PM          533.33
11          May 28 2018 11:31PM          533.33
12          Jun 28 2018 11:31PM          533.33
13          Jul 28 2018 11:31PM          533.33
14          Aug 28 2018 11:31PM          533.33
15          Sep 28 2018 11:31PM          533.33
16          Oct 28 2018 11:31PM          533.33
17          Nov 28 2018 11:31PM          533.33
```

```
18          Dec 28 2018 11:31PM          533.33
19          Jan 28 2019 11:31PM          533.33
20          Feb 28 2019 11:31PM          533.33
21          Mar 28 2019 11:31PM          533.33
22          Apr 28 2019 11:31PM          533.33
23          May 28 2019 11:31PM          533.33
24          Jun 28 2019 11:31PM          533.33
**************************************************************************
GRAND TOTAL :          12800.00
**************************************************************************
******************THANK YOU FOR VISITING OUR BANK **********************
 */

/*CURSOR*/--Pointer that points the result set --then update/delete the data--only work on result set
          --but they are very slow
          --HERE THE RESULT SET CANNOT BE UPDATED
          --BUT CURSORS CAN UPDATE THEM
          --Types of cursors

/*1)FIRE HOSE CURSOR*/
select *
from dbo.Account_Details--FIRE HOSE CURSOR
                        --THAT MEANS IT DOESNOT HAVE CURSOR
                        --contain all result set--very fast
                        --default result set
                        --only single result set

exec sp_help Account_Details
                        --it has multiple result set for single statement

/*2)SURVER SIDE  CURSOR*/
--RESULT SET is stored at server side--when you give any statement
--sql server supports it
--ex: asking ministatement at atm: ministatement is saved at survor side
--T SQL CURSORS--triggers/procedures/scripts from client




/*3)CLIENT SIDE CURSOR*/
--RESULT SET is stored at client side--when you give any statement
--sql server doesnt support it*/


/*create cursor*/

--you can use cursor any where ...procedure, function , etc
--cursor is a variable

--declare cursor cursorname cursor type
           --statement
           -- open
           --execute
           --close


declare acc_details cursor for
select* from Account_Details

open acc_details
fetch next from acc_details
fetch next from acc_details
fetch next from acc_details
fetch next from acc_details
fetch next from acc_details
fetch next from acc_details
```

```
close acc_details


deallocate acc_details


--use loop

declare engg_students cursor for
select *from ENGG_STUDENTS

open engg_students
while @@FETCH_STATUS=0
fetch next from engg_students--forward only type of cursor
close engg_students

deallocate engg_students


/*syntax
declare cursor name cursor for
statement

open cursor name
while @@fetch_status=0--if no more fetch execute
fetch next from cursor name
close cursor name

deallocate cursor name

*/

/*dir:
forward only--fetch next is used
scrollable--all fetch is allowed--go to particular row
fast forward only--top to bottom but read only
*/

/*visibility

*/

/*fetching types--this is used for the scrolling type of the cursor

fetch first
fetch next
fetch prior
fetch last
absolute n
relative n

*/

/*scrolling cursor*/
declare sales_cursor cursor for
select* from sales

open sales_cursor
fetch first from sales_cursor
fetch next from sales_cursor
fetch last from sales_cursor
fetch absolute 4 from sales_cursor
fetch relative 1 from sales_cursor
close sales_cursor

deallocate sales_cursor
```

```
--cursor is a pointer/variable store row_set--fetch a row and u/d/read but not insert--have a dir/type/ ↙
    visibility
--perform dml operations--where ever the cursor is there
--stores the result set in temp_db


/*visibility based cursor*/

--client and server visibiility each other

--scenario
-----------
--you u/d/r data at tempDB using cursor and other u/d/r/i data at db without cursor

--for static
--visibility on original table is zero
--static cursor is always read-only
--cannot perform u/d/i
--use: for making reports and see history data
--use temp db space hugely--when many people use the static cursor--good for reading and when less people

--for keyset driven
--only keys will be stored in the temp db
--you will always read the original data
--so the u/d but not other insert other person do -you can see them
--less temp db space req

--for dynamic cursors
--visibility is full

/*keyset driven cursor*/

declare policy cursor keyset for
select* from [dbo].[policy_details]

open policy
fetch last from policy
delete [dbo].[policy_details]
where current of policy

close policy
deallocate policy


/*dynamic cursor*/


declare engg cursor dynamic for
select * from [dbo].[ENGG_STUDENTS]

open engg
fetch last from engg

close engg
deallocate engg



close policy
deallocate policy


/* USER DEFINED FUNCTIONS*/
```

```
/*JUST GET THE DATA BY USING A QUERY--NO PARAMETERS/PROGRAMING/IF ELSE/FOR/--ONLY READ THE DATA---------
    VIEW
  ALL DML/DDL/CALL SP IN SP/UDF/CALL OTHER FUNCTION/USE CURSOR/PROG/CACHE EXEC PLAN AND REUSE----------SP
  NO DML(READ ONLY/STATE OF THE TABLE CANNOT BE CHANGED)/ARITHEMATIC OPERATIONS/PROG/ONLY IN NO OUT(USE
    RETURN)/CAN BE USED FOR DML --FUNCTION
  */


  --UDF

  create function job_salary
  (@id int )
  returns float
  as
  begin

  declare @salary float

  select @salary=salary as salary
  from job_salary_excel
  where id=@id
  return @salary
  end

  exec job_salary 3

  select [dbo].[job_salary](3)

select *from [dbo].[job_salary_excel]
where salary=[dbo].[job_salary](3)

--inline function

use [northwind]
go
create function order_data
(@emp_id int )
returns table
as
return(
select * from [dbo].[Orders]
where EmployeeID=@emp_id)
go

select *from [dbo].[order_data](3)


--cannot perform dml unless using variable table
select *from[dbo].[copy_sales]

create function agg_qty
(@sid int)
returns int
as
begin
declare @qty int
select @qty=sum(qty)
from [dbo].[copy_sales]
where sid=@sid
return @qty
end
go

select [dbo].[agg_qty](1) as agg_qty
```

```
--
use [AdventureWorks2012]
go
select*from [Sales].[Customer]

create function find_acc
(@cust_id int)
returns varchar(30)
as
begin
declare @acc varchar(30)
declare @i varchar(30)
select @acc=(AccountNumber like %i)
from [Sales].[Customer]
where sid=@sid
return @qty
end
go

select [dbo].[agg_qty](1) as agg_qty

--customer need to select what ever he want from the table he selected
--only selected columns
--dynamic queries
--developer writes static queries but customer uses dynamic queries


use firstprog
go
--select table to be chosen
declare @table varchar(200)
set @table ='Account_Details'

--select the column name
declare @column1 varchar(50)
set @column1='Customer_ID'

--select subcolumn to be chosen
declare @subcolumn1 varchar(50)
set @subcolumn1='3'
print 'select *from '+@table+' where '+ @column1+ ' = '+ '''+ @subcolumn1+'''' --optional
exec ('select *from '+@table+' where '+ @column1+ ' = '+''''+ @subcolumn1+'''')


--select all lines in above
use northwind
go
select * from sys.tables

use firstprog
go
--names of each table in the firstprog
select name from sys.tables
--no of tables in the firstprog
select count(*) from sys.tables
--no of rows in firstprog
select'select count(*) noofrows from '+name from sys.tables

--count of rows in each table
select count(*) noofrows from table_class
select count(*) noofrows from inventory_northwind
select count(*) noofrows from Account_Details
select count(*) noofrows from Transaction1
select count(*) noofrows from cinema_customer
select count(*) noofrows from Employee_details
select count(*) noofrows from cinema_ticket
```

```
select count(*) noofrows from sales
select count(*) noofrows from sysdiagrams
select count(*) noofrows from copy_sales
select count(*) noofrows from COLLEGE_DEPT
select count(*) noofrows from COLLEGE_STUDENT
select count(*) noofrows from MEDICAL_DOC
select count(*) noofrows from table1
select count(*) noofrows from MEDICAL_HOSPITAL
select count(*) noofrows from customer_details
select count(*) noofrows from policy_details
select count(*) noofrows from MEDICAL_SALARY
select count(*) noofrows from policy_agent_details
select count(*) noofrows from policy_enrolled_details
select count(*) noofrows from Transaction_Details
select count(*) noofrows from copy_customer_run
select count(*) noofrows from items
select count(*) noofrows from ENGG_STUDENTS
select count(*) noofrows from CONVERSION
select count(*) noofrows from SQL Server Destination
select count(*) noofrows from class_student_details
select count(*) noofrows from customer_dataflow
select count(*) noofrows from excel_oledb_conv
select count(*) noofrows from newtable_excel
select count(*) noofrows from job_salary_excel
select count(*) noofrows from Customer_bank

use AdventureWorks2012
go
select 'select count(*) as noofrows from '+name from sys.tables




--temp tables vs table variables

create table #temp1(id int, name varchar(20))
insert into #temp1(id, name) values (1, 'harish'),(2,'sunny')
select *from #temp1

--selecting some data and store in temp table
use firstprog
go
select * into #temp2
from [dbo].[Customer_bank]

select *from #temp2

--here you can select some data and then store in temp table--then again you can retrieve the data many ↙
    times with out executing query many times
--close connection and you will never see it again
--global temp ## --other query window(connection) can be executed


--variables are stored in the RAM
--cannot exec one by one line in the statement


declare @g int
set @g=10
print @g

declare @h table
(
eid int primary key,
name varchar(20)
)
```

```
insert into @h values(1, 'harish')
insert into @h values(2, 'sunny')

select *from @h


--table variables --small amount of data --faster than temp tables--works at ram
--temp tables --large data--slow than table variables


/*INDEXES*/

--FAST retrieval of data
--if table is not in order--no index/pk --heap table
--time to retrieve this is very high
--reads each row
--this is table scaning
--before creating index--items need to be in order--so indexes cannot be created in heap

SELECT *FROM HEAP_TABLE


--HEAR FIRSTLY I DIDNT USE PK THEN WHEN I USE PK IT ORDERED --IT IS BASIC INDEX--PK CREATES A CLUSTERED ↙
    INDEX
--PK sorts data--BUT while sorting it may take time


--when should i not use index
--to insert data heavily, you have not to use indexes
--oltp dont recomment index

--but OLAP dont recommend index

/*--CLUSTERED INDEX*/

SP_HELPINDEX HEAP_TABLE

--PK_HEAP_TABLE clustered, unique, primary key located on PRIMARY   EID
--indexes are nothing but table of content
--mdf/ndf files

/*
                                page(main)--root node--have index
                                1to100        1
                                101 to 200    2
                                201 to 301    3

    sub_page1      sub_page2     sub_page3            sub page4.........  branch node         indexx

    sub_sub_page1 sub_sub_page2........                                    leaf node           index

    --this is one of the clustered indexx
    --obtained by pk
    --organise the data
    --no cluster index--no pk
    --pk creates clustered index
    --one clustered index per table
    --data stored in asc order

    --WHEN TO USE CLUSTERED INDEX
    --range of rows
    --group by , order by
    --oltp
```

```
    SYNTAX FOR CREATING INDEX

    create clustered index indexname
    on table
    (column name)
*/

select *from copy_sales
create clustered index i123
    on copy_sales
    (sid)

sp_helpindex copy_sales

--i123  clustered located on PRIMARY    sid
--it can allow duplicates but pk cannot allow duplicates

select *from HEAP_TABLE
where EID<=50

--when table size is small and creating index is not useful
--if table is small--though you have index --scaning doesnot perform index scan--it goes to table scan
--if you have to retrieve high data--dont use indexes

/*NON CLUSTERED INDEX*/

--when PK created---CLUSTERED INDEX IS CREATED
--when unique key created--NON-CLUSTED INDEX IS CREATED

SELECT *from HEAP_TABLE
where CITY='CHN'

--HERE CHN MAY BE IN THE LAST PAGE --YOU DONT KNOW IF CHN EXIST OR NOT
--THIS TIME CITY ALSO NEED A INDEX--THAT IS NONCLUSTERED INDEX

create index i124
on HEAP_TABLE
(CITY)

SP_HELPINDEX HEAP_TABLE
/*
i124    nonclustered located on PRIMARY CITY
PK_HEAP_TABLE   clustered, unique, primary key located on PRIMARY   EID
*/
--CLUSTERED INDEX LEAF NODE CONTAIN DATA
--NON CLUSTERED INDEX HAS DATA SIDE TO IT

/*                                        HYD        1
                                          CHN        2
                                          SRPT       3
                                          NLG        4

                          HYD DATA     CHN DATA   SRPT DATA    NLG DATA
                          12  1          41  2     45   4      87  3              ID, POINTER TOWARDS       ↙
    CLUSTED INDEX LEAF NODE
                          76  4          65  6

--PREVIOUSLY 250 NOW 1000 NON CLUSTERED INDEX
--MANY UNIQUE KEYS FOR CREATING NON CLUSTERED INDEX

*/

/*--FRAGMENTATION

SMALL EMPTY SPACES BETWEEN THE DATA
CREATED WHEN DML ACTS ARE DONE OR MAKE ANY BULKINSERT/DELETE
```

```
HUGE FRAGMENTS --HUGE TIME TO RETRIEVE DATA
SO DEFRAGMENT THE INDEXES


  --DEFRAGMENTATION
REORGANISATION OF THE DATA


SYNTAX

DBCC INDEXDEFRAG (DB NAME, TABLE NAME)

normally used by DBA
*/

dbcc indexdefrag (firstprog, HEAP_TABLE)

/*
PK_HEAP_TABLE   1   0   0
i124            1   0   0
*/
```