How to create Database.

create database cmp(cmp is name of the Database u can give any valid name)

How to create the table.

```
create table employee
( eid int, ename varchar(25), esalary int, city varchar(25), did int
)

create table department
( deptid int, deptname varchar(25)
)

select * from employee
select * from department
```

How to insert the Records

```
insert into employee values(1,'Cmp',25000,'Mumbai',1)
insert employee values(2,'Santosh',15000,'Delhi',2)
insert into employee(eid,ename,city,esalary,did) values( 3,'Ajay','Mumbai',10000,1)
insert into employee values(4,'Shirish','Mumbai',35000,3)
insert into employee values('4','Shirish',30000,'mumbai',3)
insert into employee values(5,'Chetan')
insert into employee(eid,ename) values(5,'chetan')
insert into employee values(5,'Mihir',18000,'Chennai',4)
insert into employee values(6,'Mohan',16500,'Chennai',2)
insert into employee values(7,'Rishi',19500,'Delhi',5)
insert into employee values(8,'Pratik',55000,'Madras',4)


insert into department values(1,'IT')
insert into department values(2,'HR')
insert into department values(3,'Accounts')
insert into department values(4,'Finance')
insert into department values(5,'Sales')
insert into department values(6,'Marketing')
insert into department values(7,'Training')
```

Create table with Identity Column

```
create table emp
(
eid int identity(1,1), ename varchar(25)
)
 drop table emp
```

```
 insert into emp values('PQR')
insert  into  emp values('abc')
insert into emp values('sfhdljgjdf')
```

Creating the user-defined datatype

```
 create type disc from varchar(100) not null

create table abc
( eid int,
  empdesciption disc
)

drop table abc drop type disc

insert abc(eid) values(1)

select * from abc
```

Select Query

```
select * from employee

select eid,ename,esalary from employee

select 5+3+10 as 'TOTAL' from employee

select descs123 from employee
select descs123 as 'TOTAL' from employee

select * from employee
```

Concatinating the String Value

```
select ename +' lives in '+ city as 'StayDetails' from employee

select ename + ' is getting ' + convert(varchar(10),esalary) +
 ' salary ' as 'salarydetails' from employee

select convert(varchar,getdate(),107)
```

Calculation on Columns

```
select eid,ename,esalary/30 as 'PerDaySalary' from employee

select eid,ename,esalary*12 as 'YearlySalary' from employee
```

Display Data with User defined Column Name

Method 1

```
select eid as 'Employee ID',ename as 'Employee Name'
from employee
```

Method 2

```
select 'Employee ID '=eid,'Employee Name'=ename from employee
```

Method 3

```
select eid 'Employee ID' ,ENAME 'eMPLOYEE nAME' FROM EMPLOYEE
```

Select---Where Clause

```
select * from employee where eid=2

select * from employee where ename='CMP'

select * from employee where city='Mumbai'

select * from employee where did=2

select ename,esalary,city from employee where eid=2
```

Update records

```
update employee set city='delhi',esalary=25000 where eid=5
update employee set esalary='50000' where ename='cmp'
select * from employee
```

Distinct Keyword

```
select * from employee
select city from employee
select distinct city from employee
select distinct * from employee
```

Logical Operator (AND/OR/NOT)

```
select * from employee where
city='Mumbai' or city='Delhi'

select * from employee where city='Mumbai'  and  did=1

select * from employee where city='Mumbai' and city='delhi'

select * from employee where city='Delhi' and not city='Mumbai'
```

select * from employee where city='Delhi' or not city='Mumbai'

Relational Operator (< , >, <=, >= ,= ,!=)

select * from employee where esalary >15000

select * from employee where esalary >=15000

select * from employee where esalary < 15000

select * from employee where city='Mumbai' and esalary > 25000

select * from employee where esalary !=15000

select * from employee where city > 'MADRAS'

Range Operator (Between/ Not Between)

select * from employee where esalary between 5000 and 25000

select * from employee where esalary not between 5000 and 25000

select * from employee where city between 'DELHI' and 'Mumbai'

select * from employee where city not between 'chennai' and 'delhi'

List Operator (In/Not In)

select * from employee where city in ('Mumbai','Delhi')

select * from employee where city not in ('Mumbai','Delhi')

Like Operator (WILDCARD CHARACTERS: %, _ ,[] ,[^])

% --It Represents any sring of zero or more characters
_ --Represents a Single Character
[] --Represents any single character within the range.
[^] --Represents any single character not within the range.

select * from employee where city like 'M%'

select * from employee where ename like 's%'

select * from employee where ename like '%p'

select * from employee where city like 'M_'

select * from employee where ename like 'cm_'

```
select * from employee where ename like 'c_p'

select * from employee where  city  like  '%a_'

select * from employee where ename like 'a%y'

select * from employee where ename like 's[AB]%'

select * from employee where ename like '%o%'

select * from employee where ename like 's[^a]%'

select * from employee where ename like 's[^a-g]%'

select * from employee where ename like 's%[]'
```

IS NULL/IS NOT NULL

```
--It retrives the data where it is  null
select * from employee where did is null

select * from employee where city is not null
```

Using Aggregate Functions(avg/count/max/min/sum)

```
select sum(esalary)as 'Total' from employee

select sum(esalary)as 'Total' from employee where city='MUMBAI'

select sum(city) from employee

select count(esalary) from employee

select count(esalary) as 'Count' from employee where did=2

select count(city) as'COUNTING' from employee

select count(distinct city)from employee

select avg(esalary) from employee

select avg(esalary) from employee where city='mumbai'

select min(esalary) from employee

select min(esalary) from employee where city='chennai'

select max(esalary) from employee

select max(city) from employee

select min(city) from employee
```

Retrieving the data in Ascending or Descending order

```
-- (Default order is Ascending)
select * from employee order by esalary
select * from employee order by esalary asc

select * from employee  order  by  ename

select * from employee order by esalary desc

select * from employee where city='Mumbai' order by esalary
```

Using TOP Keyword

```
select top 3 * from employee

select top 2 * from employee where city='Mumbai'

select top 2 * from employee where city='Mumbai'
order by esalary

select top 25 percent * from employee
```

About Database and table

About Database and table
Whenever we install SQL Server Five System Databases are Automatically created.We can not drop this system defined databases.

1) Master Database contains all the system related and the server-specific configuration Information, including authorized users, system configuration settings. It also stores the initialization information of the sql server. Hence if the master database is not available the SQL database engine will not be started.

2) tempdb database is a temporary database that holds all the temporory tables and stored procedures.

3) model database acts as a template for the new databases. whenever a new database is created the contents of the model database is copied into the new database.

4) msdb database supports sql sever agent. it performs the task of backup exception handling, alert management(Errors).

5) Resourse database is a read only database that contains all the system objects such as system defined

stored procedures,system defined views.

Database files.

Each database is stored as a set of files on the hard disk of the computer.

There are three database files.

1) Primary data file

It contains the database objects. used to store user data and objects.

It has the .mdf extension.

2) Secondary data file

it also stores the database objets.database need not have secondary data files if the primary data file is large enough to hold all the data in the database.

It has .ndf extension.

3) Transction log file

It stores all the information about the Transaction(Insert,delete,update) that have ocrured in the database.

at least one transaction log file must exist for a database.

It has .ldf extension.

about database

```
sp_helpdb 'cmp'
sp_helpdb cmp

sp_helpdb
sp_spaceused
sp_renamedb 'olddbname','newdbname'

sp_renamedb 'cmp','chirag'

drop database dbname
```

about tables

```
sp_help employee
sp_help 'emp'
sp_help
sp_rename 'emp123','employee'
drop table emp

select * from employee
delete from ABC
delete from tablename where salary >50000
delete from employee where eid=8
delete emp

select * from ABC

truncate table employee
```

Alter table queries

```
--1) To Change the DataType
alter table employee
alter column eid varchar(25) not null

sp_help employee

--2)To add the Column
alter table emp
add city varchar(25)

select * from student

--3) To remove the column
alter table emp
drop column city

--4) To Change the name of the column
select * from employee
sp_rename 'employee.eid','EmpId'
sp_rename  'employee.empid','Eid'
```

Extracting Data from one table into another new table.

```
--(new table will be created automatically)
select * into abc from employee

select * from abc

drop table abc
```

Extracting data from one table and inserting into existing table

```
drop table emp
create table  emp
(
empid int,
empname varchar(25),
esalary int,
city varchar(25),
deptid int
)

insert into emp select * from employee

select * from emp
```

String Functions

```
select ascii('Abc')
select char(65)
select charindex('E','HELLO')
select left('Richard',4)
select right('Richard',4)
select  len('Richard')
select lower('RICHARD')
SELECT UPPER('richard')
select power(5,2)
select reverse('Action')
select substring('Weather',2,3)--It returns the part of the string.
--starting from 2 nd position extract three characters from the string
select sqrt(4)
select 'RICHARD' +space(2) +'hi'
select stuff('weather',2,3,'i') --It deletes the number of characters
--from the starting position (2) and deletes the specified number
--of characters (3) insert a new character(i)
```

Mathematical Function

```
select pi()
select power(2,3)
select floor(8.465)--It returns the largest Value less then or
--equal to the specified value
select log(2)

select round(1234.567,2)
select round(1234.564,2)
select round(1234.567,1)
select round(1234.467,0)
select round(1234.567,-1)
select round(1234.567,-2)
select round(1234.567,-3)
select  round(1567.567,-3)
```

Convert Function

```
select eid,convert(char(10),esalary)as 'Employee Salary'
from employee
```

Ranking Functions

```
--We can use ranking functions to give sequential numbers for each row
  or to give the ranking based on the specific criteria.

--row_number function returns the sequential numbers strating from 1
select eid,esalary,row_number() over(order by esalary desc)
as rank from employee
```

--rank function returns the rank of each row in a result based on a specified criteria.

```
select eid,esalary,rank() over(order by esalary desc)
as rank from employee
```

--dense_rank function is used to give the consecutive ranking values basedon the condition.

```
select eid,esalary,dense_rank() over(order by esalary desc)
as rank from employee
```

Date Function

Datepart	Abbrevation	Values
year	yy,yyyy	753-9999
quater	qq,q	1-4
month	mm,m	1-12
day of the year	dy,y	1-366
day	dd,d	1-31
week	wk,ww	0-51
weekday	dw	1-7 (1 is sunday)
hour	hh	0-23
minute	mi	0-59
second	ss,s	0-59
millisecond	ms	0-999

1) GETDATE()

It returns the current date and time.

```
select getdate()
```

2) DATEADD(DATEPART,NUMBER,DATE) It adds the date part to the date

```
select dateadd(dd,10,getdate())
select dateadd(mm,10,getdate())
```

3) DATEDIFF(DATEPART,DATE1,DATE2) It calculates the number of dateparts between the two dates

```
select datediff(yy,'10/23/88',getdate())
select datediff(dd,'2/10/91',getdate())
select datediff(Mi,'02/10/91',getdate())
select datediff(Mm,'02/10/91',getdate())
select datediff(wk,'02/10/91',getdate())
```

4) DATENAME(DATEPART,DATE)--It returns the datepart as a character

```
select month=Datename(mm,'01/30/87'),year=Datename(yy,'01/30/87')
select datepart(mm,getdate())
```

5) DATEPART(DATEPART,DATE) It returns the datepart as an integer

```
select datepart(mm,getdate())
```

Group By

To view data matching the specific criteria to be displayed
together in the result set we use group by clause.
It summarizes the result set into groups as defined in the query
by using aggregate function.
we can not use * in the group by clause.
the columns which is specified in the select list has to be
specified in the group by clause also..

```
select * from employee


select city,Minimum=min(esalary),Maximumm=max(esalary)
from employee
group by city

select city,Minimum=min(esalary),Maximumm=max(esalary)
from employee
where esalary>19000
group by city

select city,Minimum=min(esalary),Maximumm=max(esalary)
from employee
where esalary>19000
group by city
order by city desc
```

Group By with Having Clause.

It is same as the SELECT.....WHERE Clause.
If we want to use where clause along with aggregate function
we can not use it in group by clause so
alternate option is to use Group By with Having Clause.

```
select city,sum(esalary) from employee
group by city
having sum(esalary)>15000
```

Group By all

It is used to display all the groups including those which are
excluded by the where clause.
If all keyword is not used Group By Clause does not show the
groups for which there are no matching rows.
It displays NULL Where it does not match the records.

```
select city,Minimum=min(esalary),Maximumm=max(esalary)
from employee
where esalary>19000
group by all city
```

--CUBE Operator

```
create table sales(name varchar(30),countrycode varchar(30),sales int)
--drop table sales
select * from sales

insert into sales values('abc',001,1000)
insert into sales values('def',002,2000)
insert into sales values('abc',001,2300)
insert into sales values('def',003,3400)
insert into sales values('abc',002,1234)

Select name, countrycode, sum (sales) as totalsales from sales
Group by name, countrycode with CUBE
```

--Roll Up Operator

You won't see any difference since you're only rolling up a single column.
Consider an example where we do

ROLLUP (YEAR, MONTH, DAY)

With a ROLLUP, it will have the following outputs:

YEAR, MONTH, DAY
YEAR, MONTH
YEAR
()

With CUBE, it will have the following:

YEAR, MONTH, DAY
YEAR, MONTH
YEAR, DAY
YEAR
MONTH, DAY
MONTH
DAY
()

--CUBE essentially contains every possible rollup scenario for each node
--whereas ROLLUP will keep the hierarchy in tact (so it won't skip MONTH and show YEAR/DAY, whereas CUBE will)

```
Select name,countrycode, sum (sales) from sales
group by Name,countrycode with Rollup
```

--2nd Example OF Cube & Rollup

```
CREATE TABLE Sales1 (EmpId INT, Yr INT, Sales MONEY)

--drop table sales1

INSERT Sales1 VALUES(1, 2005, 12000)
INSERT Sales1 VALUES(1, 2006, 18000)
INSERT Sales1 VALUES(1, 2007, 25000)
INSERT Sales1 VALUES(2, 2005, 15000)
INSERT Sales1 VALUES(2, 2006, 6000)
INSERT Sales1 VALUES(3, 2006, 20000)
INSERT Sales1 VALUES(3, 2007, 24000)

SELECT EmpId, Yr, SUM(Sales) AS Sales FROM Sales1
GROUP BY EmpId, Yr WITH CUBE

SELECT EmpId, Yr, SUM(Sales) AS Sales FROM Sales1
GROUP BY EmpId, Yr WITH ROLLUP
```

--UNION AND UNION ALL

--UNION is used to select distinct values from two tables.
--UNION ALL is used to select all values (including duplicates) from the tables.
--union all is faster than union, as union involve sorting operation to find distinct records.

```
CREATE TABLE Students2000(
   Name VARCHAR(15),
   TotalMark INT)
```

```
CREATE TABLE Stundents2005(
   Name VARCHAR(15),
   TotalMark INT)


INSERT INTO Students2000 VALUES('Robert',1063)
INSERT INTO Students2000 VALUES('John',1070)
INSERT INTO Students2000 VALUES('Rose',1032)
INSERT INTO Students2000 VALUES('Abel',1002)

INSERT INTO Students2005 VALUES('Robert',1063)
INSERT INTO Students2005 VALUES('Rose',1032)
INSERT INTO Students2005 VALUES('Boss',1086)
INSERT INTO Students2005 VALUES('Marry',1034)
```

--UNION ALL

The SQL UNION ALL Operator is used to list all records from two or more select statements.
All the records from both tables must be in the same order.

```
SELECT Name,TotalMarks FROM students2000 UNION ALL
SELECT Name,TotalMarks FROM students2005
```

```
SELECT Name,TotalMarks FROM students2000 UNION
SELECT Name,TotalMarks FROM students2005
```

---INTERSECT

INTERSECT returns any distinct values that are returned by both the query on the left and right sides of the INTERSECT operand.

```
SELECT Name,TotalMarks FROM students2000 INTERSECT
SELECT Name,TotalMarks FROM students2005
```

Joins

To Retrieve the data from the multiple tables Sql server allows to use joins.
Joins allow you to view the data from the multiple tables in the single result set.

1) Inner Join

It retrives the recored from the multiple tables.
Only the rows with values satisfying the join condition will be displayed.
Rows in both the tables that do not satisfy the join condition are not displayed.
If we don't write inner keyword then the default join is Inner Join.
If we don't write inner keyword then the default join is Inner Join.

```
select e.eid,e.ename,e.esalary,e.did,d.deptname
 from employee e join department d
 on e.did=d.deptid
```

2) Outer Join

Outer Join Displays the result set containing all the rows from one table and the matching rows from the another table.

2a) Left Outer Join

A left Outer join returns all the rows from the table specified on the left side of the LEFT OUTER join keyword.
and the matching rows from the table specified on the right side.
Where it does not find the Matching record NULL will be displayed.

```
select e.*,d.deptname from employee e
left outer join department d on e.did=d.deptid
```

2b) Right Outer Join

A Right Outer join returns all the rows from the table specified on the Right side of the RIGHT OUTER join keyword.
and the matching rows from the table specified on the LEFT side.
Where it does not find the Matching record NULL will be displayed.

```
select e.*,d.* from employee e right outer join department d
 on e.did=d.deptid
```

2c) Full Outer Join

```
-- It is the combination of left and right outer join.
-- It returns all the matching and non-matching rows from both
   the tables.
-- NULL value will be displayed for the columns for which data
-- is not available.

select e.*,d.* from employee e full outer join department d
 on e.did=d.deptid
```

3) Cross Join

```
--It joins each row from one table with the each row of the other table.
--It displays the number of rows in the first table multiplied by the
-- the number of rows in the second table.
--It returns the result as an cartesian product of the two tables.

select * from employee cross join department
```

4) self join

```
--In Self join the table is joined with itself
select e1.*,e2.* from employee e1 join employee e2
 on e2.eid=e2.eid

create table employees
(
eid int,
ename varchar(25),
city varchar(25),
managerid int
)

insert into employees values(1,'cmp','Mumbai',5)
insert into employees values(2,'komal','Mumbai',5)
insert into employees values(3,'nikit','Delhi',1)
insert into employees values(4,'vishu','Mumbai',2)
insert into employees values(5,'bakri','Delhi',4)
insert into employees values(6,'kagaj','Mumbai',3)

select * from employees

select e.*,E2.ename from employees e
join employees e2
on e.managerid=e2.eid
```

7) Equi Join

```
--It is same as Inner Join.However it is using *.
--and it is used to display all the columns from both the tables.

select e.*,d.* from employee e join department d
 on e.did=d.deptid
```

SubQuery

While Querying data from the multiple tables, we might require to use the result set of one query as an input for the condition of another query.
at that time we have to use SubQuery.
SubQuery is an SQL Query that is used within another SQL Statement.
The query that represents the parent query is called as Outer query and
The query that represents the subquery is called an inner query.
In Subquery first the inner query will get Executed and
then depends on the output of the inner query outer Query will get executed.

IN Clause

```
select * from employee where did =(select
deptid from department where deptname='IT')

select * from employee where esalary >
 (select esalary from employee where ename='cmp')

select * from employee where did in(select
deptid from department where deptid>3)

select * from employee where did in(select
deptid from department where deptname='IT')

select * from employee where did in(select
deptid from department where deptid>2)
```

Exist Keyword

```
--You Can use the Subquery to check if a set of records exist.
select * from employee where exists
(select * from department where deptid=2)

--Using Aggregate functions.
select * from employee
where esalary >(select avg(esalary) from employee)
```

Constraints

It is Important to ensure that the data stored in the table
is consistent,correct and complete.
This concept of maintaining consistency,correctness and completeness
of data is called data integrity.
Data integrity is enforced to ensure that the data in the table
is accurate, consistent, and reliable.
Sql server allows you to maintain Integrity by applying
constraints and rules.
Constraints define rules and the regulation that must be
followed to maintain consistency and the correctness of data

1) Primary Key (ENTITY INTEGRITY)

It is defined on a column or a set of columns(Composite key)
whose values uniquely indentify all the rows in a table.
Those columns are known as Primarykey.
There can be only one primary key per table.
Primary key does not allow NULL Values.
By default it creates Clustered index on a column.

```
Create table abc
(
 aid int constraint pkaid primary key
)

create table  abcd
(
aid int primary key
)

insert into abcd values(2)

  alter table employee
  add constraint pkeid primary key(eid)
```

When you apply a Primary key after creating a table which has data in it.It will give you the error if the data present in that column is repeated and not unique.
It can also give you the error if the column is not marked as not null. In that case you have to execute following Query.
and alter the column as not null

```
  alter table employee
  alter column eid int not null
```

2) Unique Constraint (ENTITY INTEGRITY)

It is similar to Primary Key whose values uniquely indentify all the rows in a table.Those columns are known as Uniquekey.
You can have 249 Unique key per table.
It allows NULL value but only once (Only a single value in the entire column).
By default it creates Non-Clustered index on a column.

```
Create table abc
(
 aid int constraint pkaid unique
)

  alter table employee
  add constraint pkeid unique(eid)
```

3) Foreign Key (REFRENTAIL INTEGRITY)

You have to create Foreign key to remove the data redundancy in two tables and when the data in one table depends on the data in another table.

```
create table  employee
(
did int constraint fkdid foreign key
references department(deptid)
)

alter table department
alter column deptid int not null

alter table department
add constraint pkdid primary key(deptid)

alter table employee
add constraint fkdeptid foreign key(did)
references department(deptid)

select * from employee
select * from department

insert into employee values(12,'Santosh',55000,'mumbai',2)

delete from department where deptid=2
delete from employee where did=2

delete from department where deptid=2
```

4) Check Constraint (DOMAIN INTEGRITY)

```
--It restricts the values to be inserted in the column.
--You can define multiple check constraints on a single column.
create table employee
(
 city char(25) constraint chkcity check(city
 in('Mumbai','Delhi','Chennai','Kolkatta'))
)

alter table employee
add constraint chkcity check(city
in('Mumbai','Delhi','Chennai','Kolkatta'))

alter table employee
add constraint chkcity check(phone like('[0-9][0-9][0-9]-[0-9][0-9]
[0-9][0-9][0-9][0-9][0-9][0-9]'))

alter table employee
add constraint chksal check(esalary between 5000 and 50000)
```

5) Default Constraint (User-Defined integrity)

```
--It is used to assign a constant value to a column.
--Only one default constraint can be created for a column.
--The column should not be an identity column.
```

```
create table  employee
(
city varchar(25) default 'Mumbai'
)

alter table employee
add constraint defcity default 'Mumbai' for city
```

To remove a Constraint

```
alter table employee
drop  constraint  chkcity
```

Creating Rule

```
--It enforces domain integrity.
--It is reusable you can create it once and apply it on many columns.

create rule rultype as @cities in('Mumbai','Delhi')

sp_bindrule 'rultype','employee.city'

sp_unbindrule 'employee.city'

drop rule rultype
```

.WRITE Clause

```
create table  sample
(
description varchar(max)
)

insert into sample values('This is a very long non-unicode string')

select * from sample

update sample set description
.write('n incredibly',9,5)
```

Pivot Query

```
--It is used to convert the rows into columns.
create table cust
(custname varchar(20),
 item varchar(20),
 qty int
)
```

```
insert into cust values('SAM','MARKER',5)
insert into cust values('SAM','PENCIL',3)
insert into cust values('SAM','MARKER',3)
insert into cust values('JOHN','MARKER',1)
insert into cust values('JOHN','PENCIL',2)
insert into cust values('JOHN','MARKER',10)
insert into cust values('JOHN','PENCIL',9)


SELECT * FROM CUST
PIVOT(SUM(QTY) FOR ITEM IN ([MARKER],[PENCIL])) pvt
```

Unpivot Query

```
-- It is used to Convert columns into rows.
create table unpvttable
(
names varchar(20),
Marker int,
Pencil int
)

insert into unpvttable values('Peter',14,20)
insert into unpvttable values('John',20,10)

SELECT * FROM unpvttable
UNPIVOT(QTY FOR ITEM IN ([MARKER],[PENCIL])) unpvt
```

Indexes

1) When a user writes queries to search a record from the table which has the large amount of data.The execution time for the queries will also increase.
2) To increase the performance of the query we need to create indexes on a column.
3) There are two types of Indexes.
   a) Clustered Index: it is an index that sorts and stores the data in the table based on their key value pairs.

      Only one clustered index can be created per table.
      create the clustered index on a column that have a hich percentage unique values and are not modified often.
      whenever we create primary key clustered index will be created automatically

   b) Non-Clustered Index : Contains the index key values and row locators that point to the storage location of the data but the physical order of rows is different.
      249 non-cluster index can be created per table
      whenever we define a unique key on a table non-clustered index

will be created automatically.
create the non-clustered index on a columns whose values are not modified often.

e.g. create clustered index ix_empid
on employee(eid)
with fillfactor=10

fillfactor--It is used to reserve a percentage of free space on
--each data page of the index.

e.g. create nonclustered index ix_did
on employee(did)

Enabling or Disabling an index

alter index ix_empid
on employee disable

alter index ix_empid
on employee enable

Renaming an Index

sp_rename 'ix_empid','ix_employeeid','index'

dropping a index

drop index ix_empid on employee

VIEW

-- SQL server allows you to create views to restrict user access to the data.
--View is a virtual table which provides access to the specific columns from one or more tables.
--It is a query stored as an object in the database.
--View does not contain any data it derives the data from one or more tables called as base tables.
--It has a similar structure to the table on which the view is created.
--Apart from security reasons view can be created to retrive the data from two tables using joins. and if we need to frequently execute this quey we can create a view to execute this query.

SYNTAX:

create view viewname
with encryption
as
query
with check option

```
--with encryption option will not allow you to view the definition of
 the view.

--with check option specifies the data modification statements to meet
  the criteria given in the select statement of the view.

create view vwemps
as
select eid,esalary from employee

drop view vwemps
select * from vwemps

Create view vwemp
with encryption
As
Select e.*,d.* from employee e join department d on e.did = d.deptid

select * from vwemp
drop view vwemp

select* from employee
select * from department

Select * from vwemp

sp_help 'vwemp'

Select * from vwemp where eid = 1
Select * from vwemp where city='mumbai'

create view vwemp
as
select * from employee where did=1
with check option

select * from vwemp
update vwemp set did=3 where eid=1

select * from employee

create view vwe
as
select *from employee where city='Mumbai'

select *from vwe

update vwe set city='Delhi' where eid=1

--You can not modify did column beacuse it is used in where clause
--in view with check option
```

Modifying view

```
Alter view vwemp
As

.....(query)

---Drop a view
Drop view vwemp

--Renaming a view
SP_rename vwemp, vwemployee
```

Modifying Data using view

--View do not maintain a separate copy of data.
--The data is present in the base tables.
--Therefore you can modify the base tables by modifying the data
--in the view.

Rule

- You can not modify data in a view if the modification affects more than one base table.
- You can modify data in a view if the modification affects only one base table at a time.

```
select * from department
select * from employee

select * from vwemp

Update vwemp set city = 'Delhi' where eid = 1
Update vwemp set deptname = 'IT' where did = 2
update vwemp set city='MUMBAI',DEPTNAME='IT' where eid=3
```

Sys.sql_modules

The Sys.sql_modules system view can be used to display the view definition using a select query and supplying the object ID of the view in the where clause.

```
Select definition from Sys.sql_modules where object_id =
object_id('vwemp')

select object_id('employee')
```

Object Definition

This function can be used to display the view definition by providing the object id of the view as the input parameter.

```
Select object_definition(Object_id ('vwemps'))
```

sp_refresh view option

At the time of creating a view a view can be created with the Scheme Binding option. If this option is not used then the changes made in the base tables does not affect in the view in that case

```
drop table customer

Create table  customer
(
Custid int,
Custname varchar (50)
)

select * from customer

insert into customer values(1,'abc')

Create view vwcustomer
As
Select * from customer

Select * from vwcustomer

Alter table customer
Add age int

SELECT * FROM CUSTOMER

Select * from vwcustomer

SP_refreshview  'vwcustomer'

Select * from vwcustomer
```

Inserting data through view

```
Insert into vwcustomer values (3, 'ABC','50')
select * from customer
```

Deleting data through view

```
delete from vwcustomer where custid = 3
```

--Batches--

--A batch is a group of T-SQL statements submitted
together to the sql server for the faster execution.
--SQL server compiles all the statements of a batch into a single
executable unit called as execution plan and helps in
saving the execution time.

--To Create a batch we need to write multiple T-SQL statements
followed by the keyword GO at the end.
-- GO is a command that specifies the end of the batch and sends
the SQL Statements to the sql server.
--You can not use statements like create rule,create function,
create procedure,create trigger in a batch

```
insert
select
delete
go
```

Variables

```
declare @var int
select @var=20
--select @var
print @var

declare @var varchar(50)
select @var='cmp'
select @var as 'Variable Name'

declare @var int
declare @var1 int
declare @var2 int
select @var=5
select @var1=7
select @var2=@var+@var1
print @var2

declare @var int
declare @var1 int
declare @var2 int
select @var=50
select @var1=7
select @var2=@var*@var1
print @var2

declare @var varchar(20)
declare @var1 varchar(20)
declare @var2 varchar(20)
select @var='abc'
```

```
select @var1='pqr'
select @var2=@var+@var1
print @var2

declare @abc int
select @abc=sum(esalary)from employee
print @abc

declare @abc int
select @abc=avg(esalary)from employee
print @abc

declare @abc int
select @abc=max(esalary)from employee
select @abc
--print @abc
select @abc as 'Max Employee Salary'


declare @abc int
select @abc=count(eid)from employee
--print @abc
select @abc as 'No of Employees'
```

If-Else

```
if exists(select * from department where deptid=2)
begin
print 'The Details are'
select * from employee
end
else
print'No Records are Found'

select * from employee
```

CASE Statement

```
select eid,ename,'City Name'=
case city
when 'Mumbai' then 'The city is Mumbai'
when 'delhi' then 'The city is Delhi'
end
from employee
```

While statement

```
WHILE (SELECT AVG(esalary) FROM employee) > 5000
BEGIN
  UPDATE employee
    SET esalary = esalary+500
  IF (SELECT MAX(esalary) FROM employee) > 25000
    BREAK
  ELSE
    CONTINUE
END
PRINT 'Too much for the market to bear'
```

STORED PROCEDURE

-- Batches are temporary and to execute a batch more then once, you need to recreate sql statements and submit them to the server. this leads to an increase in the overhead as the server needs to compile and create the execution plan for the statements again therefore if you need to execute a batch multiple times you can save it within a stored procedure.

-- A stored procedure is a collection of T-SQL statements
-- and it is a precompiled object stored in the database.

--The sql server compiles the procedure and saves it as a database object.

The process of comiling a stored procedure involves the following steps.

1) The procedure is compiled and its componenets are broken into pieces. this process is known as parsing.
2) The existance of the table,view is checked this process is known as resolving.
3) The name of the stored procedure is stored in the sysobjects table and the code that creates the stored procedure is stored in the syscomments table.
4) the procedure is compiled and the blueprint for how the query will run is created.
   this blueprint is specified as execution plan. and this execution plan is saved in the procedure cache.
5) when the procedure is executed for the first time the execution plan will be read and then run. The next time the procedure is executed in the same session it will be read directly from the cache. this increase the performance of the query as the query is not compiled again.

How to create stored procedure

```
Create procedure prcemp
As
Begin
Select * from employee
End
```

SP helptext command

```
SP_helptext prcemp
```

How to execute procedure

```
Execute prcemp
Exec Prcemp
Prcemp
```

Creating OF Stored Procedure With Encryption

```
create procedure p_emp11
with encryption
as
select * from employee

sp_helptext p_emp11
```

Generic Stored Procedure (Passing Parameter)

```
--The procedure that is defined with the parameteres is known as generic
--stored procedure.
--There are two types of parameters Input and Output.
--The default parameter is input parameter.
--Input parameter takes the input from the user.
--Output parameter is used to return the values to the user.

Create procedure preemp @city varchar(15)
As
Begin
Select * from employee
Where city = @city
End

Execute preemp 'Delhi'

select * from employee
```

Output Parameter

```
create procedure prce @eid int,@ename varchar(30) output,@city
varchar(20) output
as
select @ename=ename,@city=city from employee
where eid=@eid

declare @e varchar(30),@c varchar(20)
execute prce 1,@e output,@c output
select @e,@c
select @e as 'employee name'

drop procedure prce
```

--Return Statement Inside The Stored Procedure

```
create proc calc_square @num int=0
as
begin
return(@num*@num)
end

declare @square int;
execute @square=calc_square 10;
print @square
```

Using default value

```
Create procedure prcemp1 @city varchar (15) = 'Mumbai'
as
Select * from employee
Where city = @city


prcemp1
prcemp1 'Delhi'
```

using null values

```
Create procedure preemp2
@city varchar (15) = Null
As
If @city is Null
Begin
     Print 'enter the city name'
 return
end
ELSE
      Begin
       Select * from employee where city =@city
      End

preemp2

preemp2 'Mumbai'
```

Using If-Else

```
Create procedure preemp3
as
If (select count (*) from employee where city = 'Mumbai') > 0
Begin
    Print 'Records found'
    Select * from employee where city = 'Mumbai'
 end
 Else
    Print 'Records not found'
```

Inserting data through stored procedure

```
Create procedure preemp4
@cid int, @cname varchar (20),@cage int
As
If @cname is NULL
Begin
    Print 'enter the name'
 return
End
Else
Begin
Insert into customer values (@cid, @cname,@cage)
End

Execute preemp4 1, 'ABC' ,24

select * from customer
--This way you can delete/or update also
```

Updating Record Through Stored Procedure

```
Create procedure prcupdate @id int,@name varchar(20),@age int
As
If @name is NULL
Begin
Print 'Null Value is not allowed for the column Name'
return
End
Else
Begin
update customer set custname=@name,age=@age where custid=@id
End

Execute prcupdate 1,'xyz',22
```

Modifying a stored procedure

```
Alter procedure procedurename
As
      Begin
      ……..
      End
```

Calling a Procedure from another stored procedure

```
create procedure prctemp
as
begin
execute prcemp
end

exec prctemp
```

FUNCTION

A user defined functions are database objects that contains a set of T-SQL statements, accepts parameters,performs an action and returns the result of that action as a value.

A user defined functions are used in situations where you need to implement the logic that does not involve any permanent changes to the database objects.

There are two types of Functions.
1) Creating Scalar Functions
2) Creating table-valued funtions

1) Creating Scalar Functions

Scalar functions accepts a single value as a parameter and return single value of the type specified in the RETURNS Clause

```
create function fnrate (@city varchar(25))
returns varchar(20)
as
begin
return 'Your city is ' + @city
end

declare @c varchar(25)
set @c=dbo.fnrate ('Mumbai')
print @c

create function fnrate1 (@payrate float)
returns float
as
begin
return (@payrate * 8 * 30)
end

declare @p float
set @p=dbo.fnrate1(1000)
print @p


drop function fnemps
```

TRIGGER

--Triggers are database objects which is a block of code that
constitutes a set of T-SQL statements activated in response to
certain actions such as insert, delete or update.
--Triggers are used to ensure the data integrity.
--Triggers can not returns data to the user.

There are two types of triggers
1) DML Trigger
2) DDL Trigger

DML triggers are fired when the data in the base table is affected by insert,delete or update statement.

DDL Triggers are fired in response to DDL Statements such as create table,create view etc.

They are categorised as
1) After trigger
2) instead of trigger
3) Nested Trigger

we can create more than one trigger on a single table.
then the triggers will get executed in the sequence of creation.
we can change the order of the trigger by using

```
sp_settriggerorder 'triggername','First','delete'
```

```
create table trigger_table1
(
id int,
name varchar(30),
dateofbirth datetime,
salary int
)
```

```
select * from trigger_table1
```

```
drop table trigger_table1
```

```
insert into trigger_table1  values(1,'Asdin','8/19/1984',10000)
insert into trigger_table1 values(2,'Malcolm','11/16/1984',20000)
insert into trigger_table1 values(3,'Chirag','1/1/2000',30000)
```

Creating Triggers On Insert Statement(DML)

```
create trigger trigger1
on trigger_table1
for insert
as
if (select salary from inserted)>50000
```

```
begin
print'You Cannot Insert Salary Above 50000'
rollback transaction
end

insert into trigger_table1 values(4,'Sanoj','5/5/1985',50000)
insert into trigger_table1 values(5,'Santosh','4/4/1983',50001)
```

--To View The Defination Of The Trigger

```
sp_helptext trigger1
```

--To Drop The Trigger

```
drop trigger trigger1
```

--To Alter a Trigger

```
alter trigger trigger1
on trigger_table1
with encryption
for insert
as
if (select salary from inserted)>50000
begin
print'You Cannot Insert Salary Above 50000'
rollback transaction
end
```

Creating Trigger For Updating Records (DML)

```
create trigger trigger2
on trigger_table1
for update
as
if(select dateofbirth from inserted)>getdate()
begin
print'Date OF Birth Cannot Be More Than Todays Date'
rollback transaction
end

update trigger_table1 set dateofbirth='01/15/2010' where id=4
update trigger_table1 set dateofbirth='12/31/2010'where id=4

select * from trigger_table1
```

Creating Trigger For Deleting Records (DML)

```
create trigger trigger3
on trigger_table1
for delete
as
if 'CHIRAG'in(select name from deleted)
begin
print 'You Cannot Delete The User Chirag'
rollback transaction
end

delete from trigger_table1 where name='CHIRAG'
```

Creating AFTER Trigger

```
create trigger trigger4
on  trigger_table1
after delete
as
begin
declare @num int
select @num=count(*)from deleted
print 'No of Employees Deleted = '+convert(varchar(15),@num)
end

delete from trigger_table1 where id=4

create trigger trigger5
on trigger_table1
after insert
as
begin
declare @var varchar(20)
select @var=count(*) from inserted
print'No. OF Employees Inserted=' +convert(varchar(20),@var)
end

insert into trigger_table1 values(4,'sanoj','1/1/2007',45000)
insert  into  trigger_table1  values(4,'sanoj','1/1/2007',35000)

drop trigger trigger5
```

Creating Instead OF Trigger

```
create trigger trigger5
on trigger_table1
instead of delete
as
begin
print 'You can not delete the records'
end
```

```
delete from trigger_table1 where id=4

select * from trigger_table1
```

Creating Triggers For Permission on Tables To Be Created (DDL)

```
create trigger trigger8
on database
for create_table
as
begin
print'You Do Not Have The Permission To Create a Tables'
rollback transaction
end

select * from abc
drop view abc
drop trigger trigger8
create table abc(id int,name varchar(20))
```

Creating Triggers For Permission on Views To Be Created (DDL)

```
create trigger trigger9
on database
for create_view
as
begin
print'You Do Not Have The Permission To Create Views'
rollback transaction
end

create view vabc
as
select * from trigger_table1
```

Creating Triggers For Permission on Tables To Be Altered (DDL)

```
create trigger trigger10
on database
for alter_table
as
begin
print'You Do Not Have The Permission To Alter a Table'
rollback transaction
end

alter table trigger_table1
add job varchar(20)
```

Creating Triggers For Permission on Tables To Be Deleted (DDL)

```
create trigger trigger11
on database
for drop_table
as
begin
print'You Do Not Have The Permission To Drop a Table'
rollback transaction
end

drop table trigger_table1
```

--**New Features of SQL Server 2008**

Declare and Initialize Variables

--Microsoft SQL Server® 2008 enables you to initialize variables inline as part of the variable declaration statement instead of using separate DECLARE and SET statements.
--This enhancement helps you abbreviate your code.
--With SQL Server 2005, we need to declare and initialize variables individually.

-Instead of:
--DECLARE @myVar int
--SET @myVar = 5

--you can do it in one line:. **Inline variable assignment.**

```
DECLARE @myVar int = 5
select @myVar
```

--But with SQL 2008, we can do it following way:

```
declare @temp1 int = 4,

@temp2 varchar(10)= 'Hello!'

print @t1

print @t2
```

Compound Assignment Operators

--Compound assignment operators help abbreviate code that assigns a value to a column or a variable.

```
--The new operators are:
--•       += (plus equals)
--•       -= (minus equals)
--•       *= (multiplication equals)
--•       /= (division equals)
--•       %= (modulo equals)

declare @i int
set @i = 100
set @i += 1
select @i

set @i -= 1
select @i

set @i *= 2
select @i

set @i /= 2
select @i
```

Table Value Constructor

```
--SQL Server 2008 introduces support for table value constructors
--through the VALUES clause.
--You can now use a single VALUES clause to construct a set of rows.
--One use of this feature is to insert multiple rows based on values
--in a single INSERT statement, as follows:
 DROP TABLE CUSTOMERS

CREATE  TABLE  Customers
(
  custid      INT         NOT NULL,
  companyname VARCHAR(25)  NOT  NULL,
  phone       VARCHAR(20) NOT NULL,
  address     VARCHAR(50) NOT NULL,
  CONSTRAINT PK_Customers PRIMARY KEY(custid)
)

INSERT INTO Customers
  VALUES
  (1, 'cust 1', '(111) 111-1111', 'address 1'),
  (2, 'cust 2', '(222) 222-2222', 'address 2'),
  (3, 'cust 3', '(333) 333-3333', 'address 3'),
  (4, 'cust 4', '(444) 444-4444', 'address 4'),
  (5, 'cust 5', '(555) 555-5555', 'address 5');

--Note that even though no explicit transaction is defined here,
-- this INSERT statement is considered an atomic operation.
--  So if any row fails to enter the table, the entire INSERT operation
   fails.
```