19-JUL-2021

SQL SERVER

=> SQL SERVER is a rdbms (relational database management system) product from microsoft which is used to create and to manage database.

=> SQL SERVER used for db development and administration.

Development

- creating tables
- creating views
- creating synonyms
- creating sequences
- creating indexes
- creating stored procedures
- creating stored functions
- creating triggers
- writing queries

Administration

- installation of sql server
- creating database
- creating logins
- db backup & restore
- db export & import
- db upgrdation & migration
- db mirroring & replication
- performance tuning

SQL databases :-

SQL SERVER	microsoft
ORACLE	oracle
DB2	ibm
MySQL	
POSTGRESQL	postgresql forum development

NoSQL databases :-

MongoDB
cassandra

20-JUL-21

versions of sql server :-

version	year
SQL SERVER 1.1	1991
SQL SERVER 4.2	1993
SQL SERVER 6.0	1995
SQL SERVER 6.5	1996
SQL SERVER 7.0	1998
SQL SERVER 2000	2000
SQL SERVER 2005	2005
SQL SERVER 2008	2008
SQL SERVER 2012	2012
SQL SERVER 2014	2014
SQL SERVER 2016	2016
SQL SERVER 2017	2017
SQL SERVER 2019	2019

sql server 2016 :-

1 polybase
2 json
3 temporal table to save data changes.
4 dynamic data masking and row level security

sql server 2017 :-

1 identity cache
2 New String functions
3 Automatic Tuning

sql server 2019 :-

1 Read, write, and process big data from Transact-SQL
2 Easily combine and analyze high-value relational data with high-volume big data.
3 Query external data sources.
4 Store big data in HDFS managed by SQL Server.
5 Query data from multiple external data sources through the cluster.

client/server architecture :-

1 SERVER
2 CLIENT

SERVER :-

=> server is a system where sql server software is installed and running
=> inside the server sql server manages databases
=> sql server recieves requests from client and process the requests

CLIENT :-

=> client is a sytem where users can

1 connects to server
2 submit requests to server
3 recieves response from server

client tool :-

SSMS (sql server management studio)c

21-jul-21

how to connect to sql server :-

=> to connect to sql server open SSMS and enter following details

```
server type     :-    db engine
server name     :-    WINCTRL-F9B3VH5\SQLEXPRESS
authentication  :-    sql server authentication
login           :-    SA (system admin)
```

password :- 123

creating database in sql server :-

=> in object explorer select Databases => New Database

Enter Database Name :- DB7PM

=> click OK

=> a New Database is created with following two files

1 DATA FILE (.MDF) MDF => master data file
2 LOG FILE (.LDF) LDF => log data file

=> Data File stores data and Log file stores operations

Name	Type	Size	Autoextend	path
DB7PM	DATA	8MB	64MB	C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL\DATA\
DB7PM_LOG	LOG	8MB	64MB	C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL\DATA\

command to create new database :-

=> open master database and execute the following command

```
CREATE DATABASE SALESDB
```

22-jul-21

TSQL (Transact-SQL)

=> SQL stands for structured query language.
=> it is a language used to communicate with sql server.
=> user communicates with sql server by sending commands called queries.
=> a query is a command / instruction submitted to sql server to perform some operation over db
=> SQL is originally introduced by IBM and initial name of this lang was SEQUEL and later it is renamed to SQL.
=> SQL is common to all relational databases.

SQL SERVER	ORACLE	MYSQL	POSTGRESQL	DB2
SQL	SQL	SQL	SQL	SQL

=> based on operations over db SQL is categorized into following sublanguages.

DDL (DATA DEFINITION LANGUAGE)
DML (DATA MANIPULATION LANG)
DRL (DATA RETRIEVAL LANG)
TCL (TRANSACTION CONTROL LANG)
DCL (DATA CONTROL LANG)

SQL

DDL	DML	DRL	TCL	DCL
create	insert	select	commit	grant
alter	update		rollback	revoke
drop	delete		save transaction	
truncate	merge(मिलना)			

empid ename sal data definition / metadata
100 a 9000 data

create (बनाना/generate)
alter(बदलने/change)
truncate(काट-छांट)
insert(डालने)
commit(to promise or give your loyalty)
rollback(वापस लेना/withdrow)
grant(अनुदान)
revoke(वापस लेना)

Datatypes in SQL SERVER :-

=> a datatype specifies

1 type of the data allowed
2 amount of memory should be allocated

DATATYPES

CHAR		INTEGER	FLOAT	CURRENCY	DATE	BINARY
ASCII	UNICODE	tinyint	decimal(p,s)	smallmoney	date	binary
char	nchar	smallint		money	time	varbinary
varchar	nvarchar	int			datetime	varbinary(max)
varchar(max)	nvarchar(max)	bigint				

char(size) :-

=> allows character data upto 8000 chars
=> recommeded for fixed length char columns

```
EX :-     NAME   CHAR(10)

          SACHIN----
                   wasted

          RAVI------
                  wasted
```

=> in char datatype extra bytes are wasted , so char is not recommended for variable length fields and it is recommended for fixed length fields.

```
             STATE_CODE  CHAR(2)

             AP
             TS
             MH
             UP

             COUNTRY_CODE   CHAR(3)

             IND
             USA
```

VARCHAR(SIZE) :- (var-variable , char-character)

=> allows character data upto 8000 chars
=> recommended for variable length fields

```
   ex :-   NAME   VARCHAR(10)

           SACHIN----
                   released

           RAVI------
                   released
```

=> in varchar extra bytes are releases so varchar is recommended for variable length fields

VARCHAR(MAX) :-

=> allows character data upto 2GB

NOTE :- CHAR/VARCHAR/VARCHAR(MAX) allows ascii characters (256 chars) that includes a-z,A-Z,0-9,special chars i.e. CHAR/VARCHAR/VARCHAR(MAX) allows alphanumeric data.

```
                PANNO      CHAR(10)
                PASSWORD   VARCHAR(12)
                EMAILID    VARCHAR(20)
```

NCHAR/NVARCHAR/NVARCHAR(MAX) :-

=> allows unicode characters (65536 chars) that includes ascii chars & characters belongs to different languages.

Integer Types :- (a whole number, such as 3 or 4 but not 3.5)

=> integer types allows exact/whole numbers i.e. numbers without decimal places

TINYINT	1 BYTE	0 TO 255
SMALLINT	2 BYTES	-32768 TO 32767
INT	4 BYTES	-2,147,483,648 to 2,147,483,647
BIGINT	8 BYTES	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

ex :- AGE TINYINT
EMPID SMALLINT
AADHARNO BIGINT

DECIMAL(P,S) :-

=> allows real numbers i.e. numbers with decimal part.

P => precision => total no of digits allowed
S => scale => no of digits allowed after decimal

ex :- SALARY DECIMAL(7,2)

5000
5000.50
50000.50
500000.50 => not accepted

5000.507 => accepted => 5000.51
5000.503 => accepted => 5000.50

24-jul-21

CURRENCY TYPES :-

=> currency types are used for fields related to money

SMALLMONEY	4 BYTES	- 214,748.3648 to 214,748.3647
MONEY	8 BYTES	-922,337,203,685,477.5808 to 922,337,203,685,477.5807

EX :- SALARY SMALLMONEY
BALANCE MONEY

DATE & TIME :-

DATE => allows only dates
TIME => allows only time
DATETIME => allows date & time

=> default date format in sql server is yyyy-mm-dd
=> default time format in sql server is hh:mi:ss

ex :- DOB DATE

```
          1995-10-15

          LOGIN     TIME

          10:00:00

          TXNDATE   DATETIME

          2021-07-24 11:00:00
```

BINARY TYPES :-

=> binary types allows binary data that includes audio,video,images

binary
varbinary
varbinary(max)

binary :-

=> allows binary data upto 8000 bytes
=> extra bytes are wasted

```
  ex :-  PHOTO  BINARY(5000)
```

varbinary :-

=> allows binary data upto 8000 bytes
=> extra bytes are released

```
   ex :- PHOTO  VARBINARY(5000)
```

varbinary(MAX) :-

=> allows binary data upto 2GB.

CREATING TABLES IN SQL SERVER :-

```
CREATE TABLE <TABNAME>
(
   COLNAME   DATATYPE(SIZE),
   COLNAME   DATATYPE(SIZE),
   COLNAME   DATATYPE(SIZE),
   ----------------

 )
```

Rules :-

1 tabname should start with alphabet
2 tabname should not contain spaces & special chars but allows _,#,$
3 tabname can be upto max 128 chars
4 table can have upto 1024 columns
5 table can have unlimited rows

emp123	valid
123emp	invalid
emp 123	invalid
emp*123	invalid
emp_123	valid

Example :-

create table with following structure ?

```
EMP
EMPID  ENAME   JOB     SAL     AGE     HIREDATE
```

```
CREATE TABLE EMP
(
  EMPID     SMALLINT,
  ENAME     VARCHAR(10),
  JOB       VARCHAR(10),
  SAL       SMALLMONEY,
  AGE       TINYINT,
  HIREDATE  DATE
 )
```

=> above command creates table structure that includes columns,datatype & size

SP_HELP :- command to see the structure of the table

SP => stored procedure

syn :- SP_HELP <TABNAME>

Ex :- SP_HELP EMP

```
EMPID          smallint       no      2
ENAME          varchar        no      10
JOB            varchar        no      10
SAL            smallmoney     no      4
AGE            tinyint        no      1
HIREDATE       date           no      3
```

26-jul-21

INSERTING DATA INTO TABLE :-

=> insert command is used to insert data into table
=> using insert command we can insert

1 single row

2 multiple rows

inserting single row :-

```
  syn :-  INSERT INTO <tabname> VALUES(v1,v2,v3,---)

  ex :-  INSERT INTO emp VALUES(100,'sachin','clerk',6000,40,'2021-01-01')
         INSERT INTO emp VALUES(101,'vijay','manager',8000,35,GETDATE())
```

inserting multiple rows :-

```
    INSERT INTO emp VALUES(102,'ravi','analyst',10000,40,'2020-10-05'),
                          (103,'ajay','clerk',5000,30,GETDATE())
```

inserting nulls :-

=> a NULL means blank or empty
=> it is not equal to 0 or space
=> nulls can be inserted in two ways

method 1 :-

```
      INSERT INTO emp VALUES(104,'satish',NULL,6000,NULL,GETDATE())
```

method 2 :-

```
     INSERT INTO emp(empid,ename,job,hiredate)
VALUES(105,'vinod','analyst',GETDATE())
```

=> remaining two fields sal,age filled with nulls

Displaying Data :-

=> "SELECT" command is used to display data from table.
=> using SELECT command we can display all columns or specific columns
=> using SELECT command we can display all rows or specific rows

```
    syn :- SELECT columns/*  FROM tabname


           FROM clause   => used to specify tablename
           SELECT clause => used to specify columns
                     *   => all columns

             SQL      =   ENGLISH

             queries  =   sentences

             clauses  =    words
```

=> display all the data from emp table ?

```
    SELECT * FROM emp
```

=> display employee names and salaries ?

```
    SELECT ename,sal FROM emp
```

=> display names ,jobs and hiredate ?

```
    SELECT ename,job,sal FROM emp
```

Operators in SQL SERVER :-

1 Arithmetic Operators => + - * / %

(+)Add,(-)Subtract,(*)Multiply,(/)Divide,(%)Modulo

2 Relational Operators => > >= < <= = <>

=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

3 Logical Operators => AND OR NOT

&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

4 Special Operators => BETWEEN
IN
LIKE
IS
ANY
ALL
EXISTS
PIVOT

ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

5 Set Operators => UNION
UNION ALL
INTERSECT
EXCEPT

Union	Combines distinct results of two or more SELECT statements.

Union All Combines all results of two or more SELECT statements, including duplicates.
Intersect Returns only the common records obtained from two or more SELECT statements.
Minus Returns only those records which are exclusive to the first table.

WHERE clause :-

=> used to get specific row/rows from table based on a condition

```
SELECT columns
FROM tabname
WHERE condition
```

condition :-

```
COLNAME  OPERATOR   VALUE
```

=> OPERATOR must be any relational operator like > >= <   <= = <>

=> if condition = true row is selected

=> if condition = false row is not selected

=> display employee details whose empid=103 ?

```
SELECT * FROM emp WHERE empid=103
```

=> display employee details whose name = vijay ?

```
SELECT * FROM emp WHERE ename='vijay'
```

=> display employee details earning more than 8000 ?

```
SELECT * FROM emp WHERE sal > 8000
```

=> display employee details joined after 2020 ?

```
SELECT * FROM emp WHERE hiredate > 2020  => ERROR

SELECT * FROM emp WHERE hiredate >  '2020-12-31'
```

28-jul-21

compound condition :-

=> multiple conditions combined with AND / OR operators is called compound condition

WHERE COND1	AND	COND2	RESULT
T		T	T
T		F	F
F		T	F
F		F	F

```
          WHERE COND1  OR  COND2     RESULT
                T           T        T
                T           F        T
                F           T        T
                F           F        F
```

=> display list of employee working as clerk,manager ?

```
   SELECT * FROM emp WHERE job='clerk' OR  job='manager'
```

=> display employees whose empid=100,103,105 ?

```
     SELECT * FROM emp WHERE empid=100 OR empid=103 OR empid=105
```

=> display employees working as clerk and earning more than 5000 ?

```
    SELECT * FROM emp WHERE job='clerk'  AND  sal>5000
```

=> display employees age between 30 and 40 ?

```
    SELECT * FROM emp WHERE age>=30 AND  age<=40
```

=> display employees joined in 2020 year ?

```
    SELECT * FROM emp WHERE hiredate >= '2020-01-01'  AND  hiredate <= '2020-12-31'
```

scenario :-

```
 CREATE TABLE student
 (
  sno    tinyint,
  sname  varchar(10),
  s1     tinyint,
  s2     tinyint,
  s3     tinyint
  )

 INSERT INTO student VALUES(1,'A',80,90,70),(2,'B',30,60,50)
```

```
 STUDENT
 SNO  SNAME  S1  S2  S3
 1     A     80  90  70
 2     B     30  60  50
```

=> display list of students who are passed ?

```
   SELECT * FROM student WHERE s1>=35 AND s2>=35 AND s3>=35
```

=> display list of students who are failed ?

```
   SELECT * FROM student WHERE s1<35 OR s2<35 OR s3<35
```

IN operator :-

=> use IN operator for list comparision.

```
WHERE COLNAME = V1,V2,V3   => INVALID

WHERE COLNAME IN (V1,V2,V3,--)       (WHERE COL=V1 OR COL=V2 OR COL=V3)

WHERE COLNAME NOT IN (V1,V2,V3,---)
```

=> display employees working as clerk,manager ?

```
SELECT * FROM emp WHERE job IN ('clerk','manager')
```

=> display employees whose empid=100,103,105 ?

```
SELECT * FROM emp WHERE empid IN (100,103,105)
```

=> display employees not working as clerk,manager ?

```
SELECT * FROM emp WHERE job NOT IN ('clerk','manager')
```

29-JUL-21

BETWEEN OPERATOR :-

=> use BETWEEN for range comparision

```
                                 10,20,30,40,50   LIST

                                 1O TO 50         RANGE
```

```
syn :- WHERE COLNAME BETWEEN V1 AND V2   (WHERE COL>=V1 AND COL<=V2)
       WHERE COLNAME NOT BETWEEN V1 AND V2
```

=> employees earning between 5000 and 10000 ?

```
SELECT * FROM emp WHERE sal BETWEEN 5000 AND 10000
```

=> employees whose age between 30 and 40 ?

```
SELECT * FROM emp WHERE age BETWEEN 30 AND 40
```

=> employees not joined in 2021 year ?

```
SELECT * FROM emp WHERE hiredate NOT BETWEEN '2021-01-01' AND '2021-12-31'
```

scenario :-

TRANSACTIONS

TRID	TTYPE	TDATE	TAMT	ACCNO
1	W	28-	2000	100
2	D	29-	1000	101

=> display last one week transactions of customer 100 ?

```
SELECT *
```

```
  FROM TRANSACTIONS
  WHERE TDATE BETWEEN GETDATE()-7 AND GETDATE()
        AND
        ACCNO=100
```

Question :-

```
  SELECT * FROM emp WHERE sal BETWEEN 10000 AND 5000
```

A ERROR
B RETURNS NO ROWS
C RETURN ROWS
D NONE

ANS :- B

```
     WHERE SAL BETWEEN 5000 AND 10000  (SAL>=5000 AND SAL<=10000)

     WHERE SAL BETWEEN 10000 AND 5000  (SAL>=10000 AND SAL<=5000)
```

NOTE :- use BETWEEN operator with lower and upper but not with upper and lower

=> display employees working as clerk,manager and earing between 5000 and 10000 and age between 30 and 40 and joined in 2021 year

```
   SELECT *
   FROM emp
   WHERE job IN ('clerk','manager')
         AND
         sal BETWEEN 5000 AND 10000
         AND
         age BETWEEN 30 AND 40
         AND
         hiredate BETWEEN '2021-01-01' AND '2021-12-31'
```

scenario :-

1

```
  PRODUCTS
  prodid   pname   price   category   brand
```

=> display mobiles price between 10000 and 20000 and brand = samsung,realme,redmi ?

```
   SELECT *
   FROM products
   WHERE category='mobiles'
         AND
         price between 10000 and 20000
         AND
         brand IN ('samsung','redmi','realme')
```

2

```
    CUSTOMERS
    cid    name    addr   gender   age   city   state
```

=> list of customers male living in hyd,mum,del,blr age between 25 and 40 ?

```
     SELECT *
     FROM customers
     WHERE gender='male'
           AND
           city in ('hyd','del','mum','blr')
           AND
           age BETWEEN 25 AND 40
```

30-jul-21

LIKE operator :-

=> use LIKE operator for pattern comparision.

```
   WHERE COLNAME LIKE 'PATTERN'
   WHERE COLNAME NOT LIKE 'PATTERN'
```

=> pattern contains alphabets (a-z),digits (0-9),wildcard characters

wildcard chars :-

```
   %    => 0 or many chars

   _    => exactly 1 char
```

=> display employee list name starts with 's' ?

```
   SELECT * FROM emp WHERE ename LIKE 's%'
```

=> display employee list name ends with 'd' ?

```
    SELECT * FROM emp WHERE ename LIKE '%d'
```

=> display employee list name contains 'a' ?

```
   SELECT * FROM emp WHERE ename LIKE '%a%'
```

=> display employee list where 'a' is 3rd char in their name ?

```
   SELECT * FROM emp WHERE ename LIKE '__a%'
```

=> display employee list where 'a' is 3rd char from last ?

```
      SELECT * FROM emp WHERE ename LIKE '%a__'
```

=> display employee list name contains 4 chars ?

```
   SELECT * FROM emp WHERE ename LIKE '____'
```

=> display employee joined in oct month ?

```
      yyyy-mm-dd

   SELECT * FROM emp WHERE hiredate LIKE '_____10___'
```

=> display employees joined in 2020 year ?

```
  SELECT * FROM emp WHERE hiredate LIKE '2020%'
```

Question :-

```
 SELECT * FROM emp WHERE job IN ('clerk','%man%')
```

A ERROR
B RETURNS ONLY CLERK RECORDS
C RETURNS CLERK & MANAGER RECORDS
D NONE

ANS :- B

```
 SELECT * FROM emp WHERE job IN ('clerk','%man%')
                         or
                         job LIKE '%man%'
```

ANS :- C

IS operator :-

=> use IS operator for NULL comparision

```
   WHERE COLNAME IS NULL
   WHERE COLNAME IS NOT NULL
```

=> display employees whose age = null ?

```
   SELECT * FROM emp WHERE age = NULL  => no rows

   SELECT * FROM emp WHERE age IS NULL
```

=> display employees whose age <> null ?

```
   SELECT * FROM emp WHERE age IS NOT NULL
```

summary :-

```
  WHERE COL IN (V1,V2,V3,--)
  WHERE COL BETWEEN V1 AND V2
  WHERE COL LIKE 'PATTERN'
  WHERE COL IS  NULL
```

31-jul-21

ORDER BY clause :-

=> ORDER BY clause is used to sort data based on one or more columns either in ascending order or in descending order.

```
syn :- SELECT columns
       FROM tabname
       [WHERE condition]
       ORDER BY <col> [ASC/DESC]

    ASC  => ascending
    DESC => descending
```

=> default order is ascending

Examples :-

=> arrange employee list name wise ascending order ?

```
SELECT *
FROM emp
ORDER BY ename ASC
```

=> arrange employee list sal wise desc order ?

```
SELECT *
FROM emp
ORDER BY sal DESC

1 A  3000                    5  E  6000
2 B  2000                    3  C  5000
3 C  5000  ----------->      4  D  4000
4 D  4000                    1  A  3000
5 E  6000                    2  B  2000
```

NOTE :-

=> in ORDER BY clause we can use column name or column number

```
SELECT *
FROM emp
ORDER BY 6 DESC
```

=> ORDER BY number should not be based on table , it should be based on select list

```
SELECT empno,ename,hiredate,sal,deptno
FROM emp
ORDER BY 6 DESC => ERROR

SELECT empno,ename,hiredate,sal,deptno
FROM emp
ORDER BY 4 DESC
```

=> above query sorts based on 4th column in select list i.e. sal.

=> arrange employee list dept wise asc and with in dept sal wise desc ?

```
    SELECT empno,ename,hiredate,sal,deptno
    FROM emp
    ORDER BY 5 ASC,4 DESC

    1  A  3000 20                 5  E  7000 10
    2  B  4000 10               2  B  4000 10
    3  C  6000 30 ---------->   4  D  5000 20
    4  D  5000 20               1  A  3000 20
    5  E  7000 10               3  C  6000 30
    6  F  5000 30               6  F  5000 30
```

scenario :-

STUDENTS

SID	SNAME	M	P	C
1	A	80	90	70
2	B	60	50	70
3	C	90	80	70
4	D	90	70	80

=> arrange student list total marks wise desc ,m desc,p desc ?

```
  SELECT *
  FROM student
  ORDER BY (M+P+C) DESC,M DESC,P DESC

   3  C      90     80     70
   4  D      90     70     80
   1  A      80     90     70
   2  B      60     50     70
```

02-AUG-21

=> display employees working as clerk,manager and arrange output sal wise desc order ?

```
  SELECT empno,ename,job,sal
  FROM emp
  WHERE job IN ('clerk','manager')
  ORDER BY 4 DESC
```

FROM emp :-

EMPNO	ENAME	SAL	JOB
1	A	5000	MANAGER
2	B	3000	CLERK
3	C	4000	SALESMAN
4	D	3000	CLERK
5	E	8000	MANAGER

WHERE job IN ('clerk','manager') :-

```
  1     A       5000    MANAGER
  2     B       3000    CLERK
  4     D       3000    CLERK
  5     E       8000    MANAGER
```

ORDER BY 4 DESC :-

```
 5      E       8000    MANAGER
 1      A       5000    MANAGER
 2      B       3000    CLERK
 4      D       3000    CLERK
```

```
 -----------------------------------------------------------------------------
----------
```

DML (Data Manipulation Language) commands :-

```
 --------------------------------------------
```

INSERT
UPDATE
DELETE
MERGE

```
      TABLE   =   STRUCTURE(COLUMNS)   +  DATA(ROWS)

                    DDL                   DML
```

=> all DML commands acts on table data
=> in SQL SERVER all commands are implicitly saved.
=> to turn off this auto save execute the following command

```
          SET IMPLICIT_TRANSACTIONS ON
```

=> after executing above command every operation should be explicitly saved.
=> to save the operation execute "COMMIT" command
=> to cancel the operation execute "ROLLBACK" command

UPDATE command :-

```
-----------------
```

=> command used to modify the data in a table.
=> we can update all rows or specific rows
=> we can update all columns or specific columns

```
  syn :- UPDATE tabname
         SET colname = VALUE , colname = value,-----------
         [WHERE condition]
```

=> update all employees comm with 500 ?

```
   UPDATE emp SET comm = 500
```

=> update employee comm with 500 whose empno=7369 ?

```
    UPDATE emp SET comm=500 WHERE empno=7369
```

=> update employees comm to 500 whose comm = null ?

```
UPDATE emp SET comm=500 WHERE comm IS NULL
```

=> increment sal by 20% and comm by 10% those working as salesman and joined in 1981 year ?

```
UPDATE emp
SET sal=sal+(sal*20/100),comm=comm+(comm*10/100)
WHERE job='SALESMAN'
      AND
      hiredate LIKE '1981%'
```

03-aug-21

DELETE command :-

=> command used to delete row/rows from table
=> we can delete all rows or specific rows

syn :- DELETE FROM <tabname> [WHERE cond]

=> delete all rows from emp table ?

```
DELETE FROM emp
```

=> delete employees joined in 1980 year ?

```
DELETE FROM emp WHERE hiredate LIKE '1980%'
```

=> delete employee whose empno=7844 ?

```
DELETE FROM emp WHERE empno=7844
```

=> delete employees not earning comm ?

```
DELETE FROM emp WHERE comm IS NULL
```

DDL commands :- (Data Definition Language)

create
alter
drop
truncate

=> DDL commands acts on table structure that includes columns,datatype and size

ALTER command :-

=> command used to modify the table structure
=> using ALTER command we can

```
    1 add column
    2 drop column
    3 modify a column
          1 changing size
          2 changing datatype
```

Adding a column :-

=> add column AGE to emp table ?

```
   ALTER TABLE emp
          ADD age TINYINT
```

NOTE :- after adding column ,by default the new column is filled with NULLs , so to insert data into this new column use update command.

```
  UPDATE emp SET age = 60 WHERE empno=7369
```

Droping column :-

=> drop column age from emp table ?

```
   ALTER TABLE emp
        DROP COLUMN age
```

Modifyig a column :-

=> increase the size of ename to 20 ?

```
   ALTER TABLE emp
         ALTER COLUMN ename VARCHAR(20)
```

=> decrease size of ename to 10 ?

```
      ALTER TABLE emp
         ALTER COLUMN ename VARCHAR(10)

        ALTER TABLE emp
         ALTER COLUMN ename VARCHAR(5)  => ERROR  because some names contains
                                                  more than 5 chars
```

=> change the datatype of sal column to money ?

```
    ALTER TABLE emp
        ALTER COLUMN sal MONEY

       ALTER TABLE emp
          ALTER COLUMN empno TINYINT => ERROR because values in empno column
                                              not in tinyint range
```

04-AUG-21

DROP command :-

=> command used to remove the table from database.

=> drop command drops table structure with data.

SYN :- DROP TABLE <tabname>

EX :- DROP TABLE EMP

TRUNCATE command :-

=> deletes all the data from table but keeps structure.

=> will empty the table.

=> releases memory allocated for table.

=> when we issue truncate command sql server goes to db and releases all the pages allocated for table , when pages are released , data stored in pages are also deleted.

syn :- TRUNCATE TABLE <tabname>

ex :- TRUNCATE TABLE student

DROP VS DELETE VS TRUNCATE :-

	DROP	DELETE	TRUNCATE
1	DDL command	DML command	DDL command
2	drops structure with data	deletes only data but not structure	deletes only data but not structure

DELETE VS TRUNCATE :-

	DELETE	TRUNCATE
1	DML command	DDL command
2	can delte specific row/rows	deletes only all rows but cant' delete specific rows
3	where cond can be used with delete	where cond cannot be used with truncate
4	deletes row-by-row	deletes all rows at a time
5	slower	faster
6	will not release memory	releases memory
7	will not reset identity	will reset identity

SP_RENAME :-

=> used to change tablename or column name

```
SP_RENAME 'old name','new name'
```

=> rename table EMP to EMPLOYEES ?

```
SP_RENAME 'EMP','EMPLOYEES'
```

=> rename column comm to bonus ?

```
SP_RENAME 'EMPLOYEES.COMM','BONUS'
```

IDENTITY :-

=> used to generate sequence numbers
=> used to auto increment column values

```
 syn :-   IDENTITY(SEED,INCR)

  SEED => start
          optional
          default 1

 INCR  => increment
          optional
          default 1
```

Example :-

```
CREATE TABLE cust
(
  cid     int  IDENTITY(100,1),
  cname   varchar(10)
)


INSERT INTO cust(cname) VALUES('A')
INSERT INTO cust(cname) VALUES('B')
INSERT INTO cust(cname) VALUES('C')
INSERT INTO cust(cname) VALUES('D')

SELECT * FROM cust

cid   cname
100   A
101   B
102   C
103   D
```

05-AUG-21

```
 SELECT * FROM cust                            SELECT * FROM cust

 cid   cname                                   cid     cname
 100   A                                       100     A
 101   B                                       101     B
 102   C                                       102     C
 103   D                                       103     D
 DELETE FROM cust                              TRUNCATE TABLE cust
 104    E                                      100     E
```

How to reset identity after delete ?

```
DBCC CHECKIDENT(tablename,reseed,value)

DBCC  CHECKIDENT('cust',reseed,99)

DBCC => db consistency check
```

providing explicit value for identity column :-

=> by default sql server will not allow explicit value into identity column

```
INSERT INTO cust(cid,cname) VALUES(200,'K')    => ERROR
```

=> to insert explicit value into identity column execute the following command

```
        SET IDENTITY_INSERT CUST ON

 INSERT INTO cust(cid,cname) VALUES(200,'K')  =>  1 ROW AFFECTED
```

Q can we declare identity for char fields ?

ANS :- no

```
CREATE TABLE student
(
   SID INT,
   SNAME VARCHAR(10) IDENTITY
)
```

output :- ERROR

Q can we have multiple columns declared with identity in table ?

ANS :- no

```
CREATE TABLE student
(
 SID  INT IDENTITY,
 CID  INT IDENTITY
)
```

output :- ERROR

BUILT-IN FUNCTIONS IN SQL SERVER :-

=> a function accepts some input performs some calculation and returns one value

Types of functions :-

1 date
2 string
3 mathematical
4 conversion
5 special
6 analytical
7 aggregate

DATE functions :-

1 GETDATE() :- returns current date & time

```
   SELECT GETDATE()  => 2021-08-08 19:43:39.947
```

2 DATEPART() :- used to extract part of the date

```
        DATEPART(interval,date)

   SELECT DATEPART(yy,GETDATE())         => 2021
                   mm                    => 08
                   dd                    => 05
                   hh                    => 19
                   mi                    => 50
                   ss                    => 20
                   dw                    => 05  (day of the week)

                                              01  sunday
                                              02  monday
                                              03  tuesday

                                              07  saturday
                   dayofyear             => 217 out of 365 days
                   qq                    => 3   (quarter)

                                            01 jan-mar
                                            02 apr-jun
                                            03 jul-sep
                                            04 oct-dec

                    ww                    => 32 week of the year
                    w                     => 5  week of the month
```

06-aug-21

=> display list of employees joined in 1980,1983,1985 ?

```
   SELECT * FROM emp WHERE DATEPART(yy,hiredate) IN (1980,1983,1985)
```

=> display list of employees joined on sunday ?

```
   SELECT * FROM emp WHERE DATEPART(dw,hiredate)=1
```

=> display employees joined in leap year ?

```
   SELECT * FROM emp WHERE DATEPART(yy,hiredate)%4=0
```

=> display employees joined in 2nd quarter of 1981 year ?

```
   SELECT * FROM emp WHERE DATEPART(qq,hiredate)=2
                           AND
                           DATEPART(yy,hiredate)=1981
```

DATENAME() :-

=> similar to datepart used to extract part of the date

	MM	DW
DATEPART	08	6
DATENAME	August	Friday

=> write a query to display on which day india got independence ?

```
    SELECT DATENAME(dw,'1947-08-15')   => Friday
```

=> display employees joined in jan,apr,dec months ?

```
   SELECT * FROM emp WHERE DATENAME(mm,hiredate) IN
('january','april','december')
```

DATEDIFF() :- used to find difference between two dates

```
          syn:- DATEDIFF(interval,start date,end date)

  SELECT DATEDIFF(yy,'2020-08-06',GETDATE())  => 1
  SELECT DATEDIFF(mm,'2020-08-06',GETDATE())  => 12
  SELECT DATEDIFF(dd,'2020-08-06',GETDATE())  => 365
```

=> display ENAME,EXPERIENCE in years ?

```
   SELECT ENAME,DATEDIFF(yy,hiredate,GETDATE()) as EXPR FROM emp
```

```
=> display ENAME,EXPERIENCE  ?
                 M years N months

   experience = 42 months = 3 years 6 months
```

```
        years  = months/12 = 42/12 = 3
        months = months%12 = 42%12 = 6

      SELECT ENAME,
          DATEDIFF(mm,hiredate,GETDATE())/12 AS YEARS,
          DATEDIFF(mm,hiredate,GETDATE())%12 AS MONTHS
      FROM EMP

      SMITH  40    8

        SELECT ENAME,
          CAST(DATEDIFF(mm,hiredate,GETDATE())/12  AS VARCHAR) + ' YEARS '
          + CAST(DATEDIFF(mm,hiredate,GETDATE())%12  AS VARCHAR) +  ' MONTHS
'
        FROM EMP

        SMITH  40 YEARS 8 MONTHS
```

note--

CAST()

Convert a value to an int datatype.

-CONVERT() function converts a value (of any type) into a specified datatype.

=> display employees having more than 40 years experience ?

```
     SELECT * FROM emp WHERE DATEDIFF(yy,hiredate,GETDATE()) > 40
```

07-aug-21

DATEADD() :-

=> used to add/subtract days,months,years to/from a date.

```
       DATEADD(interval,int,date)

   SELECT DATEADD(dd,10,GETDATE())    =>  2021-08-17
   SELECT DATEADD(dd,-10,GETDATE())   =>  2021-07-28
   SELECT DATEADD(mm,2,GETDATE())     =>  2021-10-7
   SELECT DATEADD(yy,1,GETDATE())     =>  2022-08-7
```

NOTE :-

```
 INSERT INTO emp(empno,ename,sal,hiredate)
VALUES(9999,'SACHIN',5000,GETDATE())
```

=> display list of employees joined today ?

```
  SELECT * FROM emp WHERE hiredate = GETDATE()  => NO ROWS

                        2021-08-07 = 2021-08-07 19:27:42.450
```

=> "=" comparision with GETDATE() always fails to overcome this problem use FORMAT function

```
     SELECT * FROM emp WHERE hiredate = FORMAT(GETDATE(),'yyyy-MM-dd')
```

2021-08-07 = 2021-08-07

Format-

shape of something or the way it is arranged.

FORMAT() ---function to format date/time values and number values.

scenario :-

GOLD_RATES

DATEID	RATE
2015-01-01	????
2015-01-02	????
2021-08-7	????

=> display todays gold rate ?

```
   SELECT RATE FROM GOLD_RATES WHERE DATEID= format(GETDATE(),'yyyy-MM-dd')
```

=> display yesterday's gold rate ?

```
   SELECT RATE FROM GOLD_RATES
               WHERE DATEID = format(DATEADD(dd,-1,GETDATE()),'yyyy-MM-dd')
```

=> display last month same day gold rate ?

```
   SELECT RATE FROM GOLD_RATES
               WHERE DATEID = format(DATEADD(mm,-1,GETDATE()),'yyyy-MM-dd')
```

=> display last year same day gold rate ?

```
SELECT RATE FROM GOLD_RATES
               WHERE DATEID = format(DATEADD(yy,-1,GETDATE()),'yyyy-MM-dd')
```

=> display list of employees joined in last 15 days ?

```
  SELECT * FROM emp
      WHERE hiredate BETWEEN DATEADD(dd,-15,GETDATE())  AND GETDATE()
```

EOMONTH() :- returns last day of the month

```
        EOMONTH(date,int)

  SELECT EOMONTH(GETDATE(),0)    => 2021-08-31
  SELECT EOMONTH(GETDATE(),1)    => 2021-09-30
  SELECT EOMONTH(GETDATE(),2)    => 2021-10-31
  SELECT EOMONTH(GETDATE(),-1)   => 2021-07-31
```

Assignment :-

1 display next month first day ?

```
SELECT DATEADD(DD,1,EOMONTH(GETDATE(),0))
```

2 display current month first day ?

```
  SELECT DATEADD(DD,1,EOMONTH(GETDATE(),-1))
```

3 display current year first day ?

```
  SELECT DATEADD(YY,DATEDIFF(yy,0,GETDATE()),0)
             OR
SELECT DATEADD(DD,1,EOMONTH(GETDATE(),-11))
```

4 display next year first day ?

```
 SELECT DATEADD(YY,DATEDIFF(yy,0,GETDATE())+1,0)
  OR
SELECT DATEADD(DD,1,EOMONTH(GETDATE(),1))
```

STRING functions :-

UPPER() :- converts string to uppercase

```
   UPPER(arg)
           string
           colname

 SELECT UPPER('hello')   => HELLO
```

LOWER() :- converts string to lowercase

```
   LOWER(arg)
           string
           colname

  SELECT LOWER('HELLO')    =>  hello
```

=> display EMPNO,ENAME,SAL ? display names in lowercase ?

```
   SELECT empno,LOWER(ename) as ename,sal FROM emp
```

=> convert names to lowercase table ?

```
   UPDATE emp SET ename = LOWER(ename)
```

09-AUG-21

LEN() :- returns string length i.e. no of chars

```
    LEN(string/colname)

  SELECT LEN('hello')          =>  5
  SELECT LEN('hello welcome')  =>  13
```

=> display employees name contains 5 chars ?

```
  SELECT * FROM emp WHERE ename LIKE '_____'
                   or)

  SELECT * FROM emp WHERE LEN(ename)=5
```

LEFT() :- used to extract part of the string starting from left .

```
  LEFT(string,len)
```

```
 SELECT LEFT('hello welcome',5)   =>   hello
```

=> display employees name starts with 's' ?

```
  SELECT * FROM emp WHERE ename LIKE 's%'

  SELECT * FROM emp WHERE LEFT(ename,1)='s'
```

scenario :-

=> generate emailids for employees ?

```
   empno     ename     emailid
   7369      smith     smi736@microsoft.com
   7499       allen     all749@microsoft.com

   SELECT empno,ename,
          LEFT(ename,3) + LEFT(empno,3) + '@microsoft.com' as emailid
   FROM emp
```

=> store emailids in db ?

STEP 1 :- add emailid column to emp table ?

```
  ALTER TABLE emp
       ADD emailid VARCHAR(30)
```

STEP 2 :- update the column with emailids ?

```
  UPDATE emp SET emailid =  LEFT(ename,3) + LEFT(empno,3) +
'@microsoft.com'
```

RIGHT() :- used to extract part of the string starting from right side.

```
        RIGHT(string,len)

  SELECT RIGHT('hello welcome',7)    =>   welcome
```

=> display employees name starts and ends with same char ?

```
   SELECT * FROM emp WHERE ename LIKE 'a%a'
                           OR
                           ename LIKE 'b%b'
                           OR
                           ename LIKE 'c%c'

    SELECT * FROM emp WHERE LEFT(ename,1)=RIGHT(ename,1)
```

SUBSTRING() :- used to extract part of the string starting from specific position

```
   SUBSTRING(string,start,len)

 SELECT SUBSTRING('hello welcome',7,4)   => welc
```

CHARINDEX() :-

=> returns position of a char in a string
=> if char found returns position , if not found returns 0

```
  CHARINDEX(char,string,[start])

 SELECT CHARINDEX('o','hello welcome')     => 5
 SELECT CHARINDEX('x','hello welcome')     => 0
 SELECT CHARINDEX('o','hello welcome',6)   => 11
 SELECT CHARINDEX('e','hello welcome',10)  => 13
```

Assignment :-

```
 CUST
 CID    CNAME
 10     sachin tendulkar
 11     virat kohli


display  CID    FNAME    LNAME   ?
         10     sachin   tendulkar

SELECT CID,
       SUBSTRING(cname,1,CHARINDEX(' ',CNAME)-1) AS FIRSTNAME,
           SUBSTRING(cname,CHARINDEX(' ',CNAME)+1,LEN(CNAME)) AS LASTNAME
FROM CUST
```

REPLACE() :- used to replace one string with another string

```
     REPLACE(str1,str2,str3)
```

=> in str1 , str2 replaced with str3

```
 SELECT REPLACE('hello','ell','abc')    =>  habco
 SELECT REPLACE('hello','l','abc')      =>  heabcabco
 SELECT REPLACE('hello','elo','abc')    =>  hello
 SELECT REPLACE('hello','ell','')       =>  ho
```

TRANSLATE() :- used to translate one char to another char

```
   TRANSLATE(str1,str2,str3)

 SELECT TRANSLATE('hello','elo','abc')   => habbc

                    e => a
                    l => b
                    o => c
```

=> TRANSLATE function can be used to encrypt data

```
    SELECT empno,ename,
             TRANSLATE(sal,'0123456789','$Kp*G%B^Q@') as sal
    FROM emp

         jones  2975   p@^%
```

=> remove all special characters from '@#he*&ll%^o$@' ?

10-aug-21

Mathematical Functions :-
----

1 abs() :- returns absolute value

```
  abs(-10)  => 10
 abs(10) ==10
abs(-10.2)==10.2
```

2 power() :- return power of two numbers

```
   power(3,2) => 9
```

3 sqrt() :- returns square root

```
   sqrt(16) => 4
```

4 square() :- returns square

```
   square(5) => 25
```

5 sign() :- used to check whether number is positive or negative

```
    sign(10)      =>  1
    sign(-10)     => -1
    sign(0)       =>  0
```

6 ROUND() :- used to round numbers to integer or to decimal places based on avg.

```
     ROUND(number,decimal places)

   SELECT ROUND(38.384675,0)   => 38

     38------------------38.5--------------------39

     number < avg   => rounded to lowest
     number >= avg  => rounded to highest

   SELECT ROUND(38.5432,0)    => 39

   SELECT ROUND(38.5462,2)    => 38.55

   SELECT ROUND(38.5432,2)    => 38.54

   SELECT ROUND(383.456,-2)   => 400

    300----------------------350----------------------400

   SELECT ROUND(383.456,-1)   => 380

    380----------------------385-------------------------390
```

```
SELECT ROUND(383.456,-3)  => 0

0-------------------------500---------------------------1000
```

=> round employee salaries to nearest hundred in table ?

```
UPDATE emp SET sal = ROUND(sal,-2)
```

CEILING() :- rounds number always to highest

```
CEILING(3.1)   => 4
```

FLOOR() :- rounded number always to lowest

```
FLOOR(3.9)     =>  3
```

Analytical functions :-

RANK & DENSE_RANK :-

=> both functions are used to calculate ranks
=> ranking is based on some column
=> for rank functions input must be sorted

```
RANK() OVER (ORDER BY COLNAME ASC/DESC)
DENSE_RANK() OVER (ORDER BY COLNAME ASC/DESC)
```

Example :-

=> display ranks of the employees based on sal and highest paid employee should get 1st rank ?

```
SELECT empno,ename,sal,
          RANK() OVER (ORDER BY sal DESC) as rnk
FROM emp

SELECT empno,ename,sal,
          DENSE_RANK() OVER (ORDER BY sal DESC) as rnk
FROM emp
```

=> diff b/w rank & dense_rank ?

1 rank function generates gaps but dense_rank will not generate gaps
2 rank function ranks may not be in sequence but in dense_rank ranks will be always in sequence

```
SAL            RNK              DRNK
5000           1                1
4000           2                2
3000           3                3
3000           3                3
3000           3                3
2000           6                4
2000           6                4
```

```
       1000          8              5
```

11-aug-21

=> display ranks of the employees based on sal ? if salarie are same then ranking should be based on experience ?

```
SELECT empno,ename,hiredate,sal,
        DENSE_RANK() OVER (ORDER BY sal DESC,hiredate ASC) as rnk
FROM emp
```

PARTITION BY clause :-

=> used to find ranks with in group , for example to find ranks with in dept first we need divide the table dept wise using PARTITION BY clause and apply dense_rank function on each partition instead of applying it on whole table.

```
SELECT deptno,empno,ename,sal,
       dense_rank() over (partition by deptno order by sal desc) as rnk
FROM emp

   10      7839    king    5000.00 1
   10      7782    clark   2450.00 2
   10      7934    miller  1300.00 3

   20      7902    ford    3000.00 1
   20      7788    scott   3000.00 1
   20      7566    jones   2975.00 2
   20      7876    adams   1100.00 3
   20      7369    smith   800.00  4

   30      7698    blake   2850.00 1
   30      7499    allen   1600.00 2
   30      7844    turner  1500.00 3
   30      7521    ward    1250.00 4
   30      7654    martin  1250.00 4
   30      7900    james   950.00  5
```

ROW_NUMBER() :-

=> returns record numbers for the records return by select query
=> row_number is also based on some column
=> row_number also accepts sorted input

```
SELECT empno,ename,sal,
          ROW_NUMBER() over (ORDER BY empno ASC) as rno
FROM emp
```

conversion functions :-

=> used to convert one datatype to another datatype

1 CAST
2 CONVERT

CAST() :-

```
CAST(source-expr as target-type)
```

```
select cast(10.5 as int)   => 10
```

display smith earns 800 ?
        allen earns 1600

```
SELECT ename + ' earns ' + CAST(sal AS VARCHAR) FROM emp
```

CONVERT() :-

```
CONVERT(target-type,source-expr)
```

```
SELECT CONVERT(INT,10.5)   => 10
```

=> using convert function we can display dates in different formats but not possible
   using cast function.

displaying dates in different formats :-

```
CONVERT(varchar,date,style-number)
```

Without century	With century (yyyy)	Standard	Input/Output (3)
1	101	U.S.	1 = mm/dd/yy 101 = mm/dd/yyyy
2	102	ANSI	2 = yy.mm.dd 102 = yyyy.mm.dd
3	103	British/French	3 = dd/mm/yy 103 = dd/mm/yyyy
4	104	German	4 = dd.mm.yy 104 = dd.mm.yyyy
5	105	Italian	5 = dd-mm-yy 105 = dd-mm-yyyy
6	106	-	6 = dd mon yy 106 = dd mon yyyy
7	107		7 = Mon dd, yy 107 = Mon dd, yyyy
8	108	-	hh:mi:ss
9	109		Default + milliseconds mon dd yyyy hh:mi:ss:mmmAM (or PM)

10	110	USA	10 = mm-dd-yy 110 = mm-dd-yyyy
11	111	JAPAN	11 = yy/mm/dd 111 = yyyy/mm/dd
12	112	ISO	12 = yymmdd 112 = yyyymmdd
13	113	Europe default + milliseconds	dd mon yyyy hh:mi:ss:mmm (24h)
14	114	-	hh:mi:ss:mmm (24h)
-	20 or 120 (2)	ODBC canonical	yyyy-mm-dd hh:mi:ss (24h)
-	21 or 25 or 121 (2)	ODBC canonical (with milliseconds) default for time, date, datetime2, and datetimeoffset	yyyy-mm-dd hh:mi:ss.mmm (24h)
22	-	U.S.	mm/dd/yy hh:mi:ss AM (or PM)
-	23	ISO8601	yyyy-mm-dd
-	126 (4)	ISO8601	yyyy-mm-ddThh:mi:ss.mmm (no spaces)

Note: For a milliseconds (mmm) value of 0, the millisecond decimal fraction value will not display. For example, the value '2012-11-07T18:26:20.000 displays as '2012-11-07T18:26:20'.

-	127(6, 7)	ISO8601 with time zone Z.	yyyy-MM-ddThh:mm:ss.fffZ (no spaces)

Note: For a milliseconds (mmm) value of 0, the millisecond decimal value will not display. For example, the value '2012-11-07T18:26:20.000 will display as '2012-11-07T18:26:20'.

-	130 (1,2)	Hijri (5)	dd mon yyyy hh:mi:ss:mmmAM

In this style, mon represents a multi-token Hijri unicode representation of the full month name. This value does not render correctly on a default US installation of SSMS.

-	131 (2)	Hijri (5)	dd/mm/yyyy hh:mi:ss:mmmAM

Money & Smallmoney formats :-

```
   CONVERT(varchar,number,style-number)
```

0 => 2 decimal places
1 => thousand seperator
2 => 4 decimal places

```
  SELECT empno,ename,CONVERT(varchar,sal,1) as sal from emp

  CREATE TABLE T(T  MONEY)
  INSERT INTO T VALUES(CONVERT(MONEY,'1,500.00',1))
  SELECT * FROM T
```

12-aug-21

special functions :-

ISNULL() :- used to convert null values

```
          ISNULL(arg1,arg2)

 if arg1=null returns arg2
 if arg1<>null returns arg1 only

SELECT ISNULL(100,200)   =>  100
SELECT ISNULL(NULL,200)   => 200
```

display ENAME,SAL,COMM,TOTSAL ?

```
     TOTSAL = SAL+COMM

SELECT ENAME,SAL,COMM,SAL+COMM AS TOTSAL FROM EMP

 smith  800   null   null
 allen  1600  300    1900

 SELECT ENAME,SAL,COMM,SAL+ISNULL(COMM,0) AS TOTSAL FROM EMP

 smith 800.00  NULL    800.00
 allen 1600.00 300.00  1900.00
```

display ENAME,SAL,COMM ? if COMM=NULL display N/A ?

```
SELECT ENAME,SAL,ISNULL(CAST(COMM AS VARCHAR),'N/A') AS COMM  FROM EMP
```

Aggregate Functions :-

=> Aggregate functions process group of rows and returns one value

MAX() :- returns maximum value

```
   MAX(arg)

 SELECT MAX(sal) FROM emp  => 5000

 SELECT MAX(hiredate) FROM emp => 2021-08-07
```

MIN() :- returns minimum value

```
  MIN(arg)

SELECT MIN(sal) FROM emp  => 800
```

SUM() :- returns total

```
 SUM(arg)

  SELECT SUM(sal) FROM emp => 39300
```

=> round total sal to thousands and display with thousand seperator ?

```
  SELECT CONVERT(VARCHAR,ROUND(SUM(SAL),-3),1)   FROM EMP  => 39,000.00

  39000----------39500--------------40000
```

AVG() :- returns average value

AVG(arg)

```
SELECT AVG(sal) FROM emp  => 2456.25
```

=> round avg(sal) to highest integer ?

```
 SELECT CEILING(AVG(SAL)) FROM EMP  => 2457
```

COUNT() :- returns no of values present in a column

```
SELECT COUNT(empno) FROM emp  => 16

SELECT COUNT(comm) FROM emp  => 4
```

note :- count function ignores nulls

COUNT(*) :- returns no of rows in a table

```
 SELECT COUNT(*) FROM emp  => 16
```

```
T1
F1
10
NULL
20
NULL
30

COUNT(F1) = 3
COUNT(*)  = 5
```

=> display how many employees joined in 1981 year ?

```
  SELECT COUNT(*) FROM emp WHERE DATEPART(yy,hiredate)=1981
```

=> display no of employees joined on sunday ?

```
   SELECT COUNT(*) FROM emp WHERE DATENAME(dw,hiredate)='sunday'
```

=> display no of employees joined in 2nd quarter of 1981 year ?

```
    SELECT COUNT(*) FROM emp WHERE DATEPART(yy,hiredate)=1981
                                   AND
                                   DATEPART(qq,hiredate)=2
```

NOTE :- GROUP functions are not allowed in where clause they are allowed only in SELECT,HAVING clauses .

```
   SELECT ename FROM emp WHERE sal= MAX(sal) ; => ERROR
```

GROUP BY clause :-

=> GROUP BY clause is used to group rows based on one or more columns to calculate
min,max,sum,avg,count for each group

```
    EMP
    EMPNO   ENAME  SAL  DEPTNO
    1       A      5000 10                                   10   13000
    2       B      6000 20 ----------GROUP BY------------->     20   15000
    3       C      7000 30                                   30   7000
    4       D      8000 10
    5       E      9000 20

       detailed data                                            summarized data
```

=> GROUP BY clause converts detailed into summarized data which is useful for analysis

syntax :-

```
 SELECT columns
 FROM tabname
 [WHERE cond]
 GROUP BY <column>
 [HAVING <cond>]
 [ORDER BY <col> ASC/DESC]
```

Execution :-

```
 SELECT
 FROM
 WHERE
 GROUP BY
 HAVING
 ORDER BY
```

=> display dept wise total salary ?

```
 SELECT deptno,SUM(sal)
 FROM emp
 GROUP BY deptno
```

FROM emp :-

```
    EMPNO   ENAME  SAL  DEPTNO
    1       A      5000 10
    2       B      6000 20
    3       C      7000 30
    4       D      8000 10
    5       E      9000 20
```

GROUP BY deptno :-

```
10

  1  A  5000
  4  D  8000
```

```
 20

   2   B  6000
   5   E  9000

 30

   3  C   7000
```

SELECT deptno,SUM(sal) :-

```
  10  13000
  20  15000
  30  7000
```

=> display job wise no of employees ?

```
  SELECT job,COUNT(*)
  FROM emp
  GROUP BY job
```

=> display no of employees joined in each year ?

```
  SELECT datepart(yy,hiredate) as year,count(*) as cnt
  FROM emp
  GROUP BY datepart(yy,hiredate)
```

=> display the departments where more than 4 employees working ?

```
 SELECT deptno,COUNT(*)
 FROM emp
 WHERE COUNT(*) > 4
 GROUP BY deptno   => ERROR
```

=> sql server calculates dept wise count after group by and it cannot calculate before group by so apply the condition COUNT(*) > 4 after group by using having clause.

```
  SELECT deptno,COUNT(*) AS CNT
  FROM emp
  GROUP BY deptno
  HAVING COUNT(*) > 4
```

16-aug-21

display job wise no of employees where job=clerk,manager and no of employees > 3 ?

```
  SELECT job,COUNT(*)
  FROM emp
  WHERE job IN ('CLERK','MANAGER')
  GROUP BY job
  HAVING COUNT(*)>3
```

WHERE VS HAVING :-

	WHERE	HAVING
1	selects specific rows	selects specific groups
2	condition can be applied without group by	cannot be applied without group by
3	conditions applied before group by	conditions applied after group by
4	use where clause if cond doesn't contain group function	use having clause if condition contains group function
5	can be used with select,update,delete commands	can be used with only select stmt and cannot be used with update,delete commands

Grouping based on multiple columns :-

=> display dept wise and with in dept job wise total salaries ?

```
 SELECT deptno,job,SUM(sal)
 FROM emp
 GROUP BY deptno,job
 ORDER BY 1 ASC

10     CLERK          1300
       MANAGER        2500
       PRESIDENT      5000

20     ANALYST        6000
       CLERK          1900
       MANAGER        3000

30     CLERK          1000
       MANAGER        2900
       SALESMAN       5700
```

Assignment :-

=> display year wise and with in year quarter wise no of employees joined ?

```
select datepart(yy,hiredate) as year,datepart(qq,hiredate)as quarter,count(*)
from emp
group by datepart(yy,hiredate),datepart(qq,hiredate)
order by 1 asc
```

ROLLUP & CUBE :-

=> rollup & cube are used to display subtotals and grand totals.

```
    GROUP BY ROLLUP(col1,col2,--)
    GROUP BY CUBE(col1,col2,----)
```

ROLLUP :-

=> ROLLUP displays subtotals for each group and also displays grand total

```
      SELECT deptno,job,SUM(sal) as totsal
      FROM emp
      GROUP BY ROLLUP(deptno,job)
      ORDER BY ISNULL(DEPTNO,99) ASC

        10     CLERK          1300.00
       10     MANAGER        2500.00
       10     PRESIDENT      5000.00
       10     NULL           8800.00 => DEPT SUBTOTAL
       20     ANALYST        6000.00
       20     CLERK          1900.00
       20     MANAGER        3000.00
       20     NULL           10900.00 => DEPT SUBTOTAL
       30     CLERK          1000.00
       30     MANAGER        2900.00
       30     SALESMAN       5700.00
       30     NULL           9600.00 => DEPT SUBTOTAL
       NULL   NULL           29300.00 => GRAND TOTAL
```

CUBE :-

=> cube displays subtotal for each group by column(deptno,job) and also displays grand total.

```
10     CLERK          1300.00
10     MANAGER        2500.00
10     PRESIDENT      5000.00
10     NULL           8800.00 => dept subtotal
20     ANALYST        6000.00
20     CLERK          1900.00
20     MANAGER        3000.00
20     NULL           10900.00 => dept subtotal
30     CLERK          1000.00
30     MANAGER        2900.00
30     SALESMAN       5700.00
30     NULL           9600.00 => dept subtotal
NULL   ANALYST        6000.00 => job subtotal
NULL   CLERK          4200.00 => job subtotal
NULL   MANAGER        8400.00 => job subtotal
NULL   PRESIDENT      5000.00 => job  subtotal
NULL   SALESMAN       5700.00 => job subtotal
NULL   NULL           29300.00 => grand total
```

Assignment :-

1

```
SALES
DATEID  PRODID  CUSTID  QTY     AMOUNT
```

=> display year wise and with in year quarter wise total amount ? display year wise subtotals ?

```
    SELECT datepart(yy,hiredate) as yy,datepart(qq,hiredate)as qq,SUM(sal) as totsal,count(*) as number
      FROM emp
      GROUP BY cube(datepart(yy,hiredate),datepart(qq,hiredate))
          order by yy
```

2

```
PERSONS
AADHARNO  NAME  GENDER  AGE  ADDR  CITY  STATE
```

=> display state wise and with in state gender wise population ? display state wise and gender wise subtotals ?

CREATING NEW TABLE FROM EXISTING TABLE :-

```
SELECT <columns>/* INTO <new-tabname> FROM <old-tabname>
```

Ex :-

```
SELECT * INTO emp10 FROM emp
```

COPYING SPECIFIC COLUMNS AND SPECIFIC ROWS :-

```
SELECT empno,ename,job,sal INTO emp11 FROM emp WHERE job IN ('clerk','manager')
```

COPYING ONLY STRUCTURE BUT NOT DATA :-

```
SELECT * INTO emp12 FROM emp WHERE 1=2
```

COPYING DATA FROM ONE TABLE TO ANOTHER TABLE

```
INSERT INTO <TARGET_TABLE>
SELECT COLUMNS/* FROM <SOURCE-TABLE>
```

ex :- copy data from emp to emp12

```
INSERT INTO emp12
SELECT * FROM emp
```

COPYING TABLE FROM ONE DB TO ANOTHER DB :-

```
select * into db8pm.dbo.cust from db7pm.dbo.cust
```

17-aug-21 Integrity Constraints

=> Integrity Constraints are the rules to maintain data integrity i.e. data quality.
=> Integrity Constraints are used to prevent users from entering invalid data.
=> Integrity Constraints are used to enforce rules like min sal must be 3000.
=> different integrity constraints in sql server

1 NOT NULL
2 UNIQUE
3 PRIMARY KEY
4 CHECK
5 FOREIGN KEY
6 DEFAULT

=> above constraints can be declared in two ways

1 column level
2 table level

column level :-

=> if constraints are declared immediately after declaring column then it is called column level

```
  CREATE TABLE <tabname>
  (
    colname datatype(size) constraint,
    colname datatype(size) constraint,
    -------------------
  )
```

NOT NULL :-

=> NOT NULL constraint doesn't accept null values
=> a column declared with NOT NULL is called mandatory column

ex :-

```
    CREATE TABLE emp11
    (
      empid  INT,
      ename  VARCHAR(10) NOT NULL
    )
```

Testing :-

```
  INSERT INTO emp11 VALUES(100,'A')  => accepted
  INSERT INTO emp11 VALUES(101,NULL) => ERROR
```

UNIQUE :-

=> UNIQUE constraint doesn't accept duplicates

ex :-

```
  CREATE TABLE emp12
  (
    empid      INT,
    ename      VARCHAR(10),
    emailid    VARCHAR(20) UNIQUE
   )
```

Testing :-

```
  INSERT INTO emp12 VALUES(100,'A','abc@gmail.com') => accepted
  INSERT INTO emp12 VALUES(101,'B','abc@gmail.com') => ERROR
  INSERT INTO emp12 VALUES(102,'C',NULL)            => accepted
  INSERT INTO emp12 VALUES(103,'D',NULL)            => ERROR
```

PRIMARY KEY :-

=> primary key doesn't accept duplicates and nulls
=> primary key is the combination of unique & not null

      primary key = unique + not null

=> table may contains no of columns but one column should be used to uniquely identify the records in table and that column should be declared with primary key.

Ex :-

```
    CREATE TABLE emp13
    (
       empid  INT PRIMARY KEY,
       ename  VARCHAR(10)
    )
```

Testing :-

```
   INSERT INTO emp13 VALUES(100,'A')   => accepted
   INSERT INTO emp13 VALUES(100,'B')   => ERROR
   INSERT INTO emp13 VALUES(null,'C')  => ERROR
```

=> PRIMARY KEY column can be used to uniquely identify the records.

=> only one primary key is allowed per table , if we want two primary keys then declare one column with primary key and another columns with unique & not null.

```
    CREATE TABLE cust
    (
      custid   INT           PRIMARY KEY  ,
      cname    VARCHAR(10)                ,
      aadharno  BIGINT         UNIQUE NOT NULL,
      panno     CHAR(10)       UNIQUE NOT NULL
     )
```

19-aug-21

CHECK constraint :-

=> use check constraint when rule based on condition.

```
    ex :-   CHECK(condition)
```

Example 1 :- SAL must be min 3000 ?

```
     CREATE TABLE emp13
     (
       empno int,
       sal   money CHECK(sal>=3000)
     )

   INSERT INTO emp13 VALUES(100,5000)
   INSERT INTO emp13 VALUES(101,1000) => ERROR
   INSERT INTO emp13 VALUES(102,NULL) => ACCEPTED
```

Example 2 :- GENDER must be 'M','F' ?

```
        GENDER CHAR(1) CHECK(GENDER IN ('M','F'))
```

Example 3 :- amt must be multiple of 100

```
        AMT  MONEY CHECK(AMT%100=0)
```

Example 4 :- pwd must be min 8 chars

```
        PWD  VARCHAR(12)  CHECK(LEN(PWD)>=8)
```

Example 5 :- emailid should contain '@'
             emailid should end with '.com' or '.co' or '.in'

```
        emailid  VARCHAR(30) CHECK(emailid LIKE '%@%'
                                   AND
                                   (
                                    emailid LIKE '%.com'
                                    OR
                                    emailid LIKE '%.co'
                                    OR
                                    emailid LIKE '%.in'
                                    ))
```

FOREIGN KEY :-

=> foreign key is used to establish relationship between two tables

```
 PROJECTS
 projid  name   duration   cost   client
 100     A      5 YEARS   100    TATA MOTORS
 101     B      3 YEARS    80     DBS
 102     C      4 YEARS   120    L&T INFRA

 EMP
 EMPID  ENAME   SAL     PROJID   REFERENCES PROJECTS(PROJID)
 1      A       5000    100
 2      B       4000    101
 3      C       6000    100
 4      D       4000    999  => NOT ACCEPTED
 5      E       3000    NULL => ACCEPTED
```

=> values entered in foreign key column should match with values entered in pk column.
=> foreign key allows duplicates and nulls.
=> after declaring foreign key a relationship is established between two tables
   called parent/child relationship.
=> pk table is parent and fk table is child.
=> by default sql server creates one to many (1:m) relationship between two tables,
   to establish one to one (1:1) relationship declare foreign key with unique constraint.

```
 CREATE TABLE projects
 (
  projid int PRIMARY KEY,
  pname  VARCHAR(10),
  duration VARCHAR(10)
 )

 INSERT INTO projects VALUES(100,'A','5 YEARS'),(101,'B','3 YEARS')

 CREATE TABLE emp_proj
 (
  empid  int PRIMARY KEY,
  ename  varchar(10) NOT NULL,
  sal    money CHECK(sal>=3000),
  projid  int  references projects(projid)
 )

 INSERT INTO emp_proj VALUES(1,'A',5000,100)
 INSERT INTO emp_proj VALUES(2,'B',4000,999)  => ERROR
 INSERT INTO emp_proj VALUES(3,'C',6000,100)
 INSERT INTO emp_proj VALUES(4,'D',3000,NULL)
```

Assignment :-

ACCOUNTS

ACCNO ACTYPE BAL

rules :-

1 accno should not be duplicate and null
2 actype must be 's' or 'c'
3 bal must be min 1000

TRANSACTIONS
TRID TTYPE TDATE TAMT ACCNO

rules :-

1 trid must be automatically generated
2 ttype must be 'w' or 'd'
3 tdate must be equal to getdate
4 tamt must be multiple of 100
5 accno should present in accounts table

Example for one to one relationship :-

PROJECTS

projid	name	client
100		
101		
102		

MANAGER

MGRNO	MNAME	PROJID
1	A	100
2	B	101
3	C	102

=> in the above example one project is managed by one manager and one manager manages one project so the relationship between two tables is one to one.

```
CREATE TABLE projects
(
 projid int PRIMARY KEY,
 pname  VARCHAR(10),
 client VARCHAR(20)
)

INSERT INTO projects VALUES(100,'A','TATA MOTORS'),(101,'B','DBS BANK')

CREATE TABLE managers
(
  mgrno  int PRIMARY KEY,
  mname  VARCHAR(10),
  projid int REFERENCES PROJECTS(PROJID) UNIQUE)
 )
```

DEFAULT :-

=> a column can be declared with default value as follows

```
     ex :-   hiredate date default getdate()
```

=> while inserting if we skip hiredate then sql server inserts default value.

```
 EX :- CREATE TABLE emp22
       (
         empno int ,
         hiredate date default GETDATE()
       )

      INSERT INTO emp22(empno) VALUES(100)
      INSERT INTO emp22 VALUES(101,'2021-01-01')
      INSERT INTO emp22 VALUES(102,null)

      SELECT * FROM emp22

      empno    hiredate
      100      2021-08-20
      101      2021-01-01
      102      null
```

TABLE LEVEL :-

=> if constraints are declared after declaring all columns then it is called table level
=> use table level to declare constraints for multiple columns or combination of columns

```
   CREATE TABLE <tabname>
   (
     COLNAME DATATYPE(size),
     COLNAME DATATYPE(size),
    -------------------------,
        CONSTRAINT(COL1,COL2,---)
  )
```

Declaring check constraint at table level :-

```
products
prodid   pname   mfd_dt        exp_dt
100      A       2021-08-20    2021-01-01   =>  INVALID
```

```
 rule :- exp_dt > mfd_dt

 CREATE TABLE products
 (
  prodid   int PRIMARY KEY,
  pname    varchar(10),
  mfd_dt   date ,
  exp_dt   date,
        CHECK(exp_dt>mfd_dt)
 )

 INSERT INTO products VALUES(100,'A',GETDATE(),'2021-01-01') => ERROR
 INSERT INTO products VALUES(101,'B','2021-01-01',GETDATE())
```

21-aug-21

composite primary key :-

=> if combination of columns uniquely identifies the records then that combination should be declared as primary key , if combination of columns declared primary key then it is called composite primary key.

=> in composite primary key combination should not be duplicate.

=> composite primary key declared at table level.

Example :-

ORDERS

ORDID	ORD_DT	DEL_DT	CID
1000			
1001			
1002			

PRODUCTS

PRODID	PNAME	PRICE
100	A	1000
101	B	2000
102	C	3000

=> to establish relationship (m:n) between orders & products then create third table in third table take primary keys of both tables as foreign keys

ORDER_DETAILS

ORDID	PRODID	QTY
1000	100	1
1000	101	1
1001	100	1
1001	101	1

=> in the above table ordid + prodid combination uniquely identifies the records so declare this combination as primary key at table level.

```
 CREATE TABLE orders
 (
   ordid  int PRIMARY KEY,
   ord_dt date,
   del_dt date,
   cid int
 )

 INSERT INTO orders VALUES(1000,getdate(),getdate()+10,10)
 INSERT INTO orders VALUES(1001,getdate(),getdate()+10,11)

CREATE TABLE products
(
 prodid  int PRIMARY KEY,
 pname   varchar(10),
 price   smallmoney
)

 INSERT INTO products VALUES(100,'A',1000),(101,'B',1500)

CREATE TABLE order_details
(
  ordid   int  REFERENCES orders(ordid),
  prodid  int  REFERENCES products(prodid),
  qty     int,
     PRIMARY KEY(ordid,prodid)
)

INSERT INTO order_details VALUES(1000,100,1)
INSERT INTO order_details VALUES(1001,100,1)
INSERT INTO order_details VALUES(1000,100,1) => ERROR
```

Which of the following constraint cannot be declared at table level ?

A UNIQUE
B CHECK
C PRIMARY KEY
D NOT NULL
E FOREIGN KEY

ANS :- D

23-aug-21

Adding constraints to existing tables :-

=> "ALTER" command is used to add constraints to existing table.

```
 CREATE TABLE emp88
 (
   eno int,
   ename varchar(10),
   sal money,
   dno int
 )
```

Adding primary key :-

=> we cannot add primary key to nullable column , to add pk first change the column
to not null then add primary key.

```
STEP 1 :-   ALTER TABLE emp88
                  ALTER COLUMN eno INT  NOT NULL

STEP 2 :-   ALTER TABLE emp88
                  ADD PRIMARY KEY(eno)
```

Adding check constraint :-

=> add check constraint with condition sal>=3000 ?

```
  ALTER TABLE emp88
      ADD  CHECK(sal>=3000)

  ALTER TABLE emp
      ADD  CHECK(sal>=3000) => ERROR
```

=> while adding constraint sql server also validates existing data. The above command returns error because in table some of the employee salaries are less than 3000 so constraint cannot be added.

WITH NOCHECK :-

=> if constraint is added with "WITH NOCHECK" option then sql server will not validate existing data it validates only new data.

```
   ALTER TABLE emp
       WITH NOCHECK ADD  CHECK(sal>=3000)
```

Adding foreign key :-

=> add foreign key to dno that should refer dept table primary key i.e. deptno

```
  ALTER TABLE emp88
       ADD FOREIGN KEY(dno) REFERENCES DEPT(deptno)
```

Droping constraints :-

```
  ALTER TABLE <TABNAME>
          DROP CONSTRAINT <NAME>
```

=> drop check constraint in emp88 table ?

```
  ALTER TABLE emp88
       DROP CONSTRAINT CK__emp88__sal__19DFD96B
```

=> drop primary key in dept table ?

```
  ALTER TABLE DEPT
      DROP  PK__DEPT__E0EB08D77A86050F  => ERROR

 DROP TABLE DEPT  => ERROR

 TRUNCATE TABLE DEPT  => ERROR
```

NOTE :-

=> primary key constraint cannot be dropped if referenced by some fk
=> primary key table cannot be dropped if referenced by some fk
=> primary key table cannot be truncated if referenced by some fk

DELETE RULES :-

1 ON DELETE NO ACTION (DEFAULT)
2 ON DELETE CASCADE
3 ON DELETE SET NULL
4 ON DELETE SET DEFAULT

=> these rules are declared with foreign key.
=> DELETE rules specifies how childs are affected if parent is deleted.

ON DELETE NO ACTION :-

=> parent row cannot be deleted if associated with child rows.

```
  CREATE TABLE dept99
   (
      dno  int primary key,
      dname varchar(10)
   )

   INSERT INTO dept99 VALUES(10,'HR'),(20,'IT')

   CREATE TABLE emp99
   (
      empno  int PRIMARY KEY,
      dno    int REFERENCES dept99(dno)
    )

   INSERT INTO emp99 VALUES(1,10),(2,10)

    DELETE FROM dept99 WHERE dno=10   => ERROR

    DELETE FROM dept99 WHERE dno=20   => 1 row affected
```

scenario :-

ACCOUNTS

ACCNO	ACTYPE	BAL
100		
101		

LOANS

ID	TYPE	AMT	ACCNO
1	H	30	100
2	C	10	100

Rule :- account closing is not possible if associated with loans

ON DELETE CASCADE :-

=> if parent row is deleted then it is deleted along with child rows.

```
CREATE TABLE dept99
(
   dno  int primary key,
   dname varchar(10)
)

INSERT INTO dept99 VALUES(10,'HR'),(20,'IT')

CREATE TABLE emp99
(
   empno  int PRIMARY KEY,
   dno    int REFERENCES dept99(dno)
               ON DELETE CASCADE
 )

INSERT INTO emp99 VALUES(1,10),(2,10)

DELETE FROM DEPT99 WHERE DNO=10 => 1 ROW AFFECTED

SELECT * FROM EMP99  => NO ROWS
```

scenario :-

```
ACCOUNTS
ACCNO   ACTYPE  BAL
100
101

TRANSACTIONS
TRID   TTYPE   TDATE   TAMT  ACCNO
1                            100
2                            100
```

RULE :- after closing account along with account delete transactions also

ON DELETE SET NULL :-

=> parent row can be deleted without deleting child rows but fk will be set to null

```
CREATE TABLE dept99
  (
     dno  int primary key,
     dname varchar(10)
  )

  INSERT INTO dept99 VALUES(10,'HR'),(20,'IT')

  CREATE TABLE emp99
  (
     empno  int PRIMARY KEY,
     dno    int REFERENCES dept99(dno)
```

```
                  ON DELETE SET NULL
     )

    INSERT INTO emp99 VALUES(1,10),(2,10)

    delete from dept99 where dno=10 => 1 row affected

    SELECT * FROM emp99

    eno   dno
    1     NULL
    2     NULL
```

scenario :-

PROJECTS

projid	name	duration
100		
101		
102		

EMP

empid	ename	projid
1		100
2		101
3		102

ON DELETE SET DEFAULT :-

=> parent row can be deleted without deleting child rows but fk will be set to default

```
  CREATE TABLE dept99
    (
       dno  int primary key,
       dname varchar(10)
    )

    INSERT INTO dept99 VALUES(10,'HR'),(20,'IT')

    CREATE TABLE emp99
    (
       empno  int PRIMARY KEY,
       dno    int DEFAULT 20
                  REFERENCES dept99(dno)
                  ON DELETE SET DEFAULT
     )

    INSERT INTO emp99 VALUES(1,10),(2,10)

    delete from dept99 where dno=10 => 1 row affected

    SELECT * FROM emp99
```

eno	dno
1	20
2	20

DISPLAY YEAR WISE AND WITH IN YEAR QUARTER WISE NO OF EMPLOYEES JOINED?

```
SELECT DATETYPE(YY,HIREDATE) ASYEAR,DATEPART(QQ,HIREDATE) AS QRT,COUNT(*)
FROM EMPLOYEESGROUP BY DATEPART(YY,HIREDATE),DATEPART(QQ,HIREDATE)
```

24-aug-21

JOINS

=> join is an operation performed to fetch data from two or more tables. -To fetch data from two tables we need to join those two tables.

=> in DB tables are normalized (divided) i.e. related data stored in multiple tables , to gather or to combine data stored in multiple tables we need to join those tables.

Example :-

ORDERS

ordid	ord_dt	del_dt	cid
1000	10-	20-	10
1001	11-	21-	11

CUSTOMERS

cid	cname	caddr
10	a	hyd
11	b	hyd

output :-

ordid	ord_dt	del_dt	cname	caddr
1000	10-	20-	a	hyd

Types of joins :-

1 Equi Join / Inner Join
2 Outer join
    left join
    right join
    full join
3 Non Equi Join
4 Self join
5 Cross join / Cartesian join

Equi Join :-

=> to perform equi join between the tables there must be a common field and name of the common field need not to be same and pk-fk relationship is not compulsory.

=> equi join is performed between the tables sharing common field with same datatype.

```
  SELECT columns
  FROM tabname
  WHERE join condition
```

join condition :-

=> based on the given join condition sql server joins the records of two tables.

=> join condition decides which record of 1st table should be joined with which record of second table.

```
     table1.commonfield = table2.commonfield
```

=> this join is equi join because here join condition based on "=" operator

Example :-

EMP

EMPNO	ENAME	SAL	DEPTNO
7369	smith	920.00	20
7499	allen	1920.00	30
7521	ward	1500.00	30
7566	jones	3421.25	20
7654	martin	1500.00	30
7698	blake	3420.00	30
7782	clark	2695.00	10

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTS	NEW YORK
20	RESEARCH	
30	SALES	
40	OPERATIONS	

```
create table emp
(
empno int,
ename varchar(10),
sal money,
deptno int,
)

insert into emp
values(7369,'smith',920.00,20),(7698,'blake',3420.00,30),(7782,'clark',2695.00,10),
(7499,'allen',1920.00,30),(7521,'ward',1500.00,30),(7566,'jones',3421.25,20),
(7654,'martin',1500.00,30)

create table dept
(
deptno int,
dname varchar(10),
loc varchar(10),
)
insert into dept values(10,'ACCOUNTS','NEW YORK'),(20,'RESEARCH','hyd'),(30,'SALES','bbsr'),(40,'OPERATIONS','cuttack')
```

```
display  EMPNO  ENAME  SAL  DNAME   LOC  ?
         ----------------- -------------
                EMP           DEPT

SELECT empno,ename,sal,dname,loc
 FROM  emp,dept
WHERE  emp.deptno = dept.deptno
```

```
display  EMPNO  ENAME  SAL  DEPTNO  DNAME   LOC  ?
          ----------------- -------------
                 EMP           DEPT

 SELECT empno,ename,sal,
        deptno,dname,loc
  FROM  emp,dept
 WHERE  emp.deptno = dept.deptno  => ERROR
```

=> in join queries declare table alias(उपनाम) and prefix(ex-sub name/last name -my is paul) column names with table alias for
two reasons

1 to avoid ambiguity(possibility of being understood in more than one way)
2 for faster execution

```
 SELECT e.empno,e.ename,e.sal,
        d.deptno,d.dname,d.loc
FROM  emp as e,dept as d
 WHERE  e.deptno = d.deptno
```

25-aug-21

=> display employee details with dept details working at NEW YORK loc ?

```
 SELECT e.empno,e.ename,e.sal,
        d.deptno,d.dname,d.loc
FROM  emp as e,dept as d
 WHERE  e.deptno = d.deptno /* join cond */
        AND
        d.loc='NEW YORK'  /* fiter cond */
```

joining more than 2 tables :-

=> when no of tables increases no of join conditions also increases
=> to join N tables N-1 join conditions required.

```
   SELECT columns
   FROM tab1,
        tab2,
        tab3,
        -----
   WHERE cond1
         AND
         cond2
         AND
```

```
          ------
 Example :-


  EMP         DEPT       LOCATIONS        COUNTRIES
  empno       deptno     locid            country_id
  ename       dname      city             country_name
  sal         locid      state
  deptno                 country_id

create table emp ===emp created
(
empno int,
ename varchar(10),
sal money,
deptno int,
)

insert into emp
values(7369,'smith',920.00,20),(7698,'blake',3420.00,30),(7782,'clark',2695.0
0,10),
(7499,'allen',1920.00,30),(7521,'ward',1500.00,30),(7566,'jones',3421.25,20),
(7654,'martin',1500.00,30)

create table dept==dept created
(
deptno int,
dname varchar(10),
locid varchar(10),
)
insert into dept values(10,'ACCOUNTS','NEW
YORK'),(20,'RESEARCH','hyd'),(30,'SALES','bbsr'),(40,'OPERATIONS','cuttack')

3-
create table LOCATIONS
(
locid int,
city varchar(10),
state varchar(10),
country_id char(3),
)
insert into LOCATIONS values(30,'NEW
YORK','us','usa'),(40,'hyd','telengana','ind'),(50,'bbsr','odisha','ind'),(60
,'cuttack','odisha','ind')
4-
create table countries
(
country_id char(3),
country_name varchar(10),
)
insert into countries values('usa','amarica'),('ind','india')



 => Display   ENAME    DNAME    CITY   STATE   COUNTRY_NAME  ?
              -----    -----    ------------   -------------
              EMP      DEPT      LOCATIONS      COUNTRIES
```

```
SELECT e.ename,
       d.dname,
       l.city,l.state,
       c.country_name
FROM   emp e,
       dept d,
       locations l,
       countries c
WHERE  e.deptno = d.deptno
       AND
       d.locid = l.locid
       AND
       l.country_id = c.country_id
```

=> we can write join queries in 2 styles

1 NATIVE STYLE (SQL SERVER)
2 ANSI STYLE

ANSI style :-

=> Advantage of ANSI style is portability.
=> join queries becomes portable i.e. queries can migrated from one db to another db.
=> in ANSI style tablenames are seperated by keywords
=> use ON clause for join conditions instead of WHERE clause

display ENAME DNAME ?

```
SELECT e.ename,d.dname
  FROM emp e INNER JOIN dept d
    ON e.deptno = d.deptno
```

display ENAME DNAME working at NEW YORK loc ?

```
SELECT e.ename,d.dname
  FROM emp e INNER JOIN dept d
    ON e.deptno = d.deptno
 WHERE d.loc='NEW YORK'
```

NOTE :- use ON clause for join conditions
        use WHERE clause for filter conditions

how to join multiple tables in ANSI style :-

```
SELECT columns
FROM tab1 INNER JOIN tab2
  ON condition
          INNER JOIN tab3
  ON  condition
 --------------
```

```
  EMP         DEPT       LOCATIONS        COUNTRIES
  empno       deptno     locid            country_id
  ename       dname      city             country_name
  sal         locid      state
  deptno                 country_id

  DISPLAY  ENAME   COUNTRY_NAME   ?

  SELECT   e.ename,
           c.country_name
   FROM    emp e INNER JOIN dept d
     ON    e.deptno = d.deptno
                 INNER JOIN locations l
     ON    d.locid = l.locid
                 INNER JOIN countries c
     ON    l.country_id = c.country_id
```

27-aug-21

OUTER JOIN :-

=> equi join returns only matching rows but cannot return unmatched rows but to get unmatched rows also perform outer join.

Example :-

```
  EMP                                        DEPT
  EMPNO ENAME SAL  DEPTNO                    DEPTNO  DNAME        LOC
  1     A     5000 10                      10      ACCOUNTS
  2     B     3000 20                        20      RESEARCH
  3     C     4000 30                        30      SALES
  4     D     2000 20                        40      OPERATIONS => unmatched row
  5     E     3000 NULL => unmatched row
```

=> outer join is possible in ANSI style.

=> outer join is 3 types

1 left join
2 right join
3 full join

left join :-

=> returns all rows (matched + unmatched) from left side table and matching rows from right side table.

```
  SELECT e.ename,d.dname
    FROM emp e LEFT JOIN dept d
      ON e.deptno = d.deptno
```

=> above query returns all rows from emp table and matching rows from dept table.

```
   A   ACCOUNTS
   B   RESEARCH
   C   SALES
   D   RESEARCH
   E   NULL    => unmatched row from emp
```

RIGHT JOIN :-

=> returns all rows from right side table and matching rows from left side table.

```
    SELECT e.ename,d.dname
      FROM emp e RIGHT JOIN dept d
        ON e.deptno = d.deptno

   A   ACCOUNTS
   B   RESEARCH
   C   SALES
   D   RESEARCH
  NULL OPERATIONS => unmatched from dept table
```

FULL JOIN :-

=> returns all rows from both tables

```
    SELECT e.ename,d.dname
     FROM emp e FULL JOIN dept d
       ON e.deptno = d.deptno

   A   ACCOUNTS
   B   RESEARCH
   C   SALES
   D   RESEARCH
   E   NULL     => unmatched row from emp
 NULL  OPERATIONS => unmatched row from dept
```

scenario :-

PROJECTS

projid	name	duration	cost	client
100				
101				
102				

EMP

empid	ename	sal	projid
1			100
2			101
3			100
4			null

=> display employee details with project details and also display employees not assigned to any project ?

```
 SELECT e.*,p.*
  FROM  emp e LEFT JOIN  projects p
    ON  e.projid = p.projid
```

=> display employee details with project details and also display projects where no employee assigned to it ?

```
  SELECT e.*,p.*
  FROM  emp e RIGHT JOIN projects p
    ON  e.projid = p.projid
```

Displaying unmatched rows :-

display unmatched row from left side table ?

```
   SELECT e.ename,d.dname
      FROM emp e LEFT JOIN dept d
        ON e.deptno = d.deptno
     WHERE d.dname IS NULL
```

display unmatched row from right side table ?

```
   SELECT e.ename,d.dname
      FROM emp e RIGHT JOIN dept d
        ON e.deptno = d.deptno
     WHERE e.ename IS NULL
```

display unmatched rows from both tables ?

```
   SELECT e.ename,d.dname
      FROM emp e FULL JOIN dept d
        ON e.deptno = d.deptno
     WHERE e.ename IS NULL   OR   d.dname IS NULL
```

28-AUG-21

NON EQUI JOIN :-

=> non equi join is performed between the two tables not sharing a common field.

=> this join is called non equi join because here join condition is not based "=" operator and it is based on > < between operators.

Example :-

```
  EMP                          SALGRADE
  EMPNO  ENAME  SAL            GRADE  LOSAL   HISAL
  1      A      5000            1      700     1000
  2      B      2500            2      1001    2000
  3      C      1000            3      2001    3000
  4      D      3000            4      3001    4000
  5      E      1500            5      4001    9999
```

=> display EMPNO ENAME   SAL   GRADE ?

```
    SELECT e.empno,e.ename,e.sal,s.grade
     FROM  emp e,salgrade s
    WHERE  e.sal BETWEEN s.losal and s.hisal

    1  A   5000  5
    2  B   2500  3
    3  C   1000  1
    4  D   3000  3
    5  E   1500  2
```

ANSI style :-

```
     SELECT e.empno,e.ename,e.sal,s.grade
     FROM  emp e JOIN salgrade s
       ON  e.sal BETWEEN s.losal and s.hisal
```

=> display grade 3 employee list ?

```
   SELECT e.empno,e.ename,e.sal,s.grade
     FROM  emp e JOIN salgrade s
       ON  e.sal BETWEEN s.losal and s.hisal
    WHERE  s.grade = 3
```

```
=> display  ENAME   DNAME    GRADE  ?
            ------  -----   ------
              EMP   DEPT     SALGRADE
```

```
    SELECT e.ename,d.dname,s.grade
     FROM  emp e INNER JOIN dept d
       ON  e.deptno = d.deptno
                 JOIN salgrade s
       ON  e.sal BETWEEN s.losal and s.hisal
```

SELF JOIN :-

=> joining a table to itself is called self join.
=> in self join a record in one table joined with another record of same table.

Example :-

```
        EMP
       EMPNO   ENAME   MGR
       7369    SMITH   7902
       7499    ALLEN   7698
       7521    WARD    7698
       7566    JONES   7839
```

```
        7654    MARTIN  7698
        7698    BLAKE   7839
        7782    CLARK   7839
        7788    SCOTT   7566
        7839    KING    NULL
        7902    FORD    7566
```

=> above table contains MGR but to display manager name we need to perform self join.

=> to perform self join the same table must be declared two times different alias

```
                    FROM emp X,emp Y

         EMP X                                        EMP Y

        EMPNO   ENAME   MGR                           EMPNO   ENAME   MGR
        7369    SMITH   7902                          7369    SMITH   7902
        7499    ALLEN   7698                          7499    ALLEN   7698
        7521    WARD    7698                          7521    WARD    7698
        7566    JONES   7839                          7566    JONES   7839
        7654    MARTIN  7698                          7654    MARTIN  7698
        7698    BLAKE   7839                          7698    BLAKE   7839
        7782    CLARK   7839                          7782    CLARK   7839
        7788    SCOTT   7566                          7788    SCOTT   7566
        7839    KING    NULL                          7839    KING    NULL
        7902    FORD    7566                          7902    FORD    7566
```

=> display ENAME MGRNAME ?

```
    SELECT X.ENAME,Y.ENAME AS MANAGER
    FROM emp X,emp Y
    WHERE X.MGR = Y.EMPNO
```

ANSI style :-

```
    SELECT X.ENAME,Y.ENAME AS MANAGER
    FROM emp X JOIN emp Y
      ON X.MGR = Y.EMPNO
```

=> display employees reporting to blake ?

```
    SELECT X.ENAME,Y.ENAME AS MANAGER
     FROM emp X JOIN emp Y
       ON X.MGR = Y.EMPNO
    WHERE Y.ENAME='BLAKE'
```

=> display blake's manager name ?

```
    SELECT X.ENAME,Y.ENAME AS MANAGER
     FROM emp X JOIN emp Y
       ON X.MGR = Y.EMPNO
    WHERE X.ENAME='BLAKE'
```

=> display employees earning more than their managers ?

```
     SELECT X.ENAME,Y.ENAME AS MANAGER
     FROM emp X JOIN emp Y
       ON X.MGR = Y.EMPNO
    WHERE X.SAL > Y.SAL
```

30-AUG-21

Question :-

TEAMS

ID	COUNTRY
1	IND
2	AUS
3	RSA

=> write a query to display following output ?

```
  IND VS AUS
  IND VS RSA
  AUS VS RSA
```

CROSS JOIN :-

=> cross join returns cross product of two tables

A=1,2
B=3,4

AXB = (1,3) (1,4) (2,3) (2,4)

=>if we perform cross join between two tables then each record of 1st table joined
with each and every record of second table.

=> to perform cross join(,) submit the join query without join condition

```
    SELECT e.ename,d.dname
    FROM emp e,dept d
```

ANSI STYLE :-

```
    SELECT e.ename,d.dname
    FROM emp e CROSS JOIN dept d
```

CREATING NEW FROM THE QUERY OUTPUT :-

```
 SELECT E.EMPNO,E.ENAME,E.SAL,
        D.DEPTNO,D.DNAME,D.LOC  INTO EMP_DEPT
 FROM EMP E INNER JOIN DEPT D
   ON E.DEPTNO = D.DEPTNO
```

GROUP BY & JOIN :-

=> display dept wise total sal ? display dept names in output ?

```
 SELECT d.dname,SUM(e.sal) as totsal
   FROM emp e INNER JOIN dept d
     ON e.deptno = d.deptno
 GROUP BY d.dname
```

scenario :-

SALES

DATEID	PRODID	CUSTID	QTY	AMOUNT
2021-08-30	100	10	1	3000

PRODUCTS

PRODID	PNAME	PRICE	CATEGORY
100			ELECTRONICS

CUSTOMERS

CUSTID	NAME	ADDR	CITY	STATE	COUNTRY

```
create table sales
(
dateid date,
prodid int,
custid int,
qty tinyint,
amount money,
)

create table products
(
prodid int,
pname varchar(10),
price money,
catagory varchar(15),
)

create table customers
(
custid int,
name varchar(10),
addr varchar(15),
city varchar(10),
state varchar(10),
country varchar(10),
)
```

=> display year wise total sales amount ?
=> display category wise total sales amount ?
=> display country wise total sales amount ?
=> display year wise,country wise,category wise total sales amount ?

```
 SELECT datepart(yy,s.dateid) as year,
        c.country ,
        p.category,SUM(s.amount) as total
 FROM sales s INNER JOIN customers c
   ON s.custid = c.custid
```

```
              INNER JOIN products p
   ON s.prodid = p.prodid
 GROUP BY datepart(yy,s.dateid),
        c.country ,
        p.category
```

UPDATE command & join :-

CUST1			CUST2		
CID	NAME	CITY	CID	NAME	CITY
1	A	HYD	1	A	HYD
2	B	CHE	2	B	DEL
3	C	BLR	3	C	MUM

=> update cust2 city field with cust1 table city field ?

```
  UPDATE CUST2
     SET CUST2.city = cust1.city
  FROM cust2 INNER JOIN cust1
   ON cust2.cid = cust1.cid
```

-------SET command is used with UPDATE to specify which columns and values that should be updated in a table.

31-aug-21

SET operators :-

UNION
UNION ALL
INTERSECT
EXCEPT

A=1,2,3,4
B=1,2,5,6

```
A UNION B     =  1,2,3,4,5,6
A UNION ALL B =  1,2,3,4,1,2,5,6
A INTERSECT B =  1,2
A EXCEPT B    =  3,4
B EXCEPT A    =  5,6
```

=> in SQL SERVER set operations are performed between output of two select statements

=> these operations are performed between set of rows return by two select statement

```
 SELECT STATEMENT 1
 UNION / UNION ALL / INTERSECT / EXCEPT
 SELECT STATEMENT 2
```

Rules :-
----

 1 no of columns return by both queries must be same
 2 corresponding columns datatype must be same

```
SELECT job  FROM emp WHERE deptno=20

CLERK
MANAGER
ANALYST
CLERK
ANALYST

SELECT job FROM emp WHERE deptno=30

SALESMAN
SALESMAN
SALESMAN
MANAGER
SALESMAN
CLERK
```

UNION :-

=> combines rows return by two select statements
=> eliminates duplicates
=> sorts result

```
SELECT job FROM emp WHERE deptno=20
UNION
SELECT job FROM emp WHERE deptno=30

ANALYST
CLERK
MANAGER
SALESMAN

SELECT job,sal FROM emp WHERE deptno=20
UNION
SELECT job,sal FROM emp WHERE deptno=30

ANALYST        3000.00
CLERK          950.00
CLERK          1100.00
CLERK          2000.00
MANAGER        2850.00
MANAGER        2975.00
SALESMAN       1250.00
SALESMAN       1500.00
SALESMAN       1600.00
```

union vs join :-

	union	join
1	horizontal merge	vertical merge

```
        2    combines rows                      combines columns

        3    performed between two              performed between two
dissimilar structures
              similar structures


 T1             T2
 F1             C1
 1              10
 2              20
 3              30

T1 U T2 :-

 1
 2
 3
 10
 20
 30

 T1 JOIN T2 :-

 1      10
 2      20
 3      30

scenario :-
-----------

EMP_US
ENO     ENAME   SAL     DNO

                                                      DEPT
EMP_IND                                               DNO     DNAME   LOC
ENO     ENAME   SAL     DNO


 => display total employee list ?

  SELECT * FROM EMP_US
  UNION
  SELECT * FROM EMP_IND
  UNION
  SELECT * FROM DEPT

 => dsiplay employees working at US loc with dept details ?

   SELECT E.*,D.*
   FROM EMP_US E INNER JOIN DEPT D
    ON E.DEPTNO = D.DEPTNO

 => display total employee with dept details ?

  SELECT E.*,D.*
```

```
   FROM EMP_US E INNER JOIN DEPT D
    ON E.DEPTNO = D.DEPTNO
  UNION
 SELECT E.*,D.*
   FROM EMP_IND E INNER JOIN DEPT D
    ON E.DEPTNO = D.DEPTNO
```

01-sep-21

UNION ALL :-

=> combines rows return by two select statements
=> duplicates are not eliminated
=> result is not sorted

```
SELECT job FROM emp WHERE deptno=20
UNION ALL
SELECT job FROM emp WHERE deptno=30
```

```
CLERK
MANAGER
ANALYST
CLERK
ANALYST
SALESMAN
SALESMAN
SALESMAN
MANAGER
SALESMAN
CLERK
```

=> diff b/w UNION & UNION ALL ?

	UNION	UNION ALL
1	eliminates duplicates	includes duplicates
2	sorts result	doesn't sort result
3	slower	faster

INTERSECT :-

=> returns common values from the output of two select statements

```
 SELECT job FROM emp WHERE deptno=20
 INTERSECT
 SELECT job FROM emp WHERE deptno=30
```

```
 CLERK
 MANAGER
```

EXCEPT :-

=> returns values present in 1st query output and not present in 2nd query output

```
SELECT job FROM emp WHERE deptno=20
EXCEPT
SELECT job FROM emp WHERE deptno=30
```

```
 ANALYST
```

```
SELECT job FROM emp WHERE deptno=30
EXCEPT
SELECT job FROM emp WHERE deptno=20
```

```
SALESMAN
```

Question :-

T1	T2
F1	C1
1	1
2	2
3	3
10	40
20	50
30	60

=> write the output for the following operations ?

1 INNER JOIN
2 LEFT JOIN
3 RIGHT JOIN
4 FULL JOIN
5 UNION
6 UNION ALL
7 INTERSECT
8 EXCEPT

SUB-QUERIES / NESTED QUERIES :-

=> a query in another another query is called sub-query or nested query
=> one query is called inner/child/sub-query
=> other query is called outer/parent/main query
=> first sql server executes inner query and output of inner query is input to outer query
=> use subquery when where condition based on unknown value.

Types of Subqueries :-
----

1 single row subqueries
2 multi row subqueries
3 co-related subqueries
4 Derived tables
5 Scalar subquereis

single row subqueries :-

=> if inner query returns one value then subquery is called single row subquery

syntax :-

```
      SELECT columns
      FROM tabname
      WHERE colname OP (SELECT statement)
```

=> display employees earning more than blake ?

```
    SELECT *
    FROM emp
    WHERE sal > (SELECT sal FROM emp WHERE ename='blake')
```

=> display employees who are senior to king ?

```
   SELECT *
   FROM emp
   WHERE hiredate < (SELECT hiredate FROM emp WHERE ename='king')
```

=> display employee name earning max salary ?

```
   SELECT ename
   FROM emp
   WHERE sal = MAX(sal)  => ERROR
```

=> aggregates like min,max,sum,avg,count are not allowed in where clause and they are allowed only in select,having clause.

```
   SELECT ename
   FROM emp
   WHERE sal = (SELECT MAX(sal) FROM emp)
```

=> display employee name having max experience ?

```
   SELECT ename
   FROM emp
   WHERE hiredate = (SELECT MIN(hiredate) FROM emp)
```

NOTE :- outer query can be SELECT/INSERT/UPDATE/DELETE but inner query must be always SELECT.

=> increment sal by 10% having max experience ?

```
   UPDATE emp
   SET sal=sal+(sal*0.1)
   WHERE hiredate = (SELECT MIN(hiredate) FROM emp)
```

=> delete the employee having max experience ?

```
   DELETE FROM emp WHERE hiredate = (SELECT MIN(hiredate) FROM emp)
```

02-sep-21

multi-row subqueries :-

=> if subquery returns more than one value then it is called multirow subquery

```
   SELECT columns
   FROM tabname
   WHERE colname OP (SELECT STATEMENT)
```

=> OP must be IN,NOT IN,ANY,ALL

Example :-

=> display list of employees whose job=job of smith,blake ?

```
   SELECT *
   FROM emp
   WHERE job IN (
                  SELECT job
                  FROM emp
                  WHERE ename IN ('smith','blake')
                )
```

ANY operator :-

=> used for comparision with any value i.e. atleast one

```
   WHERE X > ANY(1000,2000,3000)

   IF X=800   FALSE
      X=1500  TRUE
      X=4500  TRUE

  WHERE X < ANY(1000,2000,3000)

  IF X=800    TRUE
     X=1500   TRUE
     X=4500   FALSE
```

ALL operator :-

=> used for comparision with all values.

```
   WHERE X>ALL(1000,2000,3000)

    IF X=800    FALSE
       X=1500   FALSE
       X=4500   TRUE

   WHERE X<ALL(1000,2000,3000)

   IF X=800  TRUE
```

```
      X=1500 FALSE
      X=4500 FALSE
```

single	multi
=	IN
>	>ANY >ALL
<	<ANY <ALL

=> display employees earning more than all managers ?

```
   SELECT *
   FROM emp
   WHERE sal >   ALL (SELECT sal
                       FROM emp
                      WHERE job='MANAGER')
```

=> display employees earning more than atleast one managers ?

```
   SELECT *
   FROM emp
   WHERE sal >   ALL (SELECT sal
                       FROM emp
                      WHERE job='MANAGER')
```

co-related subqueries :-

=> if inner query refers values of outer query then it is called co-related subquery.

=> here execution starts from outer query and inner query is executed for each row
return by outer query.

=> use co-related subquery to execute subquery for each row return by outer query

1 returns a row from outer query
2 pass value to inner query
3 executes inner query
4 returns value to outer query
5 executes outer query where cond

03-sep-21

Example :-

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	10
2	B	4000	20
3	C	7000	30

```
4       D       8000    20
5       E       4000    10
```

=> display employee list earning more than avg(sal) of their dept ?

```
  SELECT *
  FROM emp x
  WHERE sal > (SELECT AVG(sal)
               FROM emp
               WHERE deptno = x.deptno)
```

```
 1      A       5000    10    5000 > (SELECT AVG(sal) FROM emp WHERE deptno=10)
4500  T
 2      B       4000    20    4000 > (SELECT AVG(sal) FROM emp WHERE deptno=20)
6000  F
 3      C       7000    30    7000 > (SELECT AVG(sal) FROM emp WHERE deptno=30)
7000  F
 4      D       8000    20    8000 > (SELECT AVG(sal) FROM emp WHERE deptno=20)
6000  T
 5      E       4000    10    4000 > (SELECT AVG(sal) FROM emp WHERE deptno=10)
4500  F
```

=> display employees earning maximum sal in their dept ?

```
  SELECT *
  FROM emp x
  WHERE sal =  (SELECT MAX(sal)
               FROM emp
               WHERE deptno = x.deptno)
```

=> display top 3 maximum salaries ?

```
  SELECT DISTINCT a.sal
  FROM emp a
  WHERE 3 > (SELECT COUNT(DISTINCT b.sal)
             FROM emp b
             WHERE a.sal < b.sal)
 ORDER BY sal DESC
```

```
   emp a              emp b
  SAL                SAL
  5000               5000    3 > (0)  TRUE
  1000                1000    3 > (4)  FALSE
  3000                3000    3 > (2)  TRUE
  2000                2000    3 > (3)  FALSE
  4000                4000    3 > (1)  TRUE
```

=> display 5th max salary ?

```
  SELECT DISTINCT a.sal
  FROM emp a
  WHERE (5-1) =  (SELECT COUNT(DISTINCT b.sal)
                  FROM emp b
                  WHERE a.sal < b.sal)
 ORDER BY sal DESC
```

EXISTS operator :-

=> use EXISTS operator to check whether record exists in the table or not

```
  SELECT
  FROM tabname
  WHERE EXISTS (SELECT STATEMENT)
```

=> EXISTS returns

```
 TRUE    => if subquery returns atleast one row
 FALSE   => if subquery returns 0 rows
```

Example :-

PRODUCTS

PRODID	PNAME	PRICE	CATEGORY
100			
101			
102			

ORDERS

ORDID	PRODID	QTY
1000	100	1
1000	101	2
1001	100	1

=> display list of products which are ordered by customer ?

method 1 :-

```
  SELECT *
  FROM products p
  WHERE EXISTS (SELECT * FROM orders WHERE prodid = p.prodid)
```

method 2 :-

```
  SELECT *
  FROM products
  WHERE prodid IN (SELECT prodid FROM orders)
```

=> SQL SERVER recommends EXISTS operator instead of IN operator because EXISTS gives good performance than IN operator.

05-sep-21

Derived Tables :-

=> subqueries in FROM clause are called derived tables.

```
     SELECT columns/*
     FROM (SELECT STATEMENT)  <ALIAS>
     [WHERE COND]
```

=> subquery output acts like a table for outer query

=> derived tables are used in the following scenarios

  1 to control order of execution of clauses
  2 to join query output with a table
  3 to use the result of one operation in another operation

=> by default sql server executes the clauses in the following order

```
 FROM
 WHERE
 GROUP BY
 HAVING
 SELECT
 ORDER BY
```

=> use derived tables to control this order of execution

Example 1 :-

=> display ENAME ANNUAL SALARY ?

```
 SELECT ename,sal*12 as annsal
 FROM emp
```

=> above query displays annual salaries of all the employees but to display employees whose annual sal > 30000

```
   SELECT ename,sal*12 as annsal
   FROM emp
   WHERE annsal > 30000  => ERROR
```

=> column alias cannot be referenced in where clause because where clause is executed before select , to overcome this proble use derived tables.

```
   SELECT *
   FROM (SELECT ename,sal*12 as annsal
         FROM emp)  E
   WHERE annsal>30000
```

Example 2 :-

```
  SELECT empno,ename,sal,
      DENSE_RANK() OVER (ORDER BY SAL DESC) AS RNK
  FROM EMP
```

=> above query display ranks of all the employees but to display top 5 employees

```
  SELECT empno,ename,sal,
      DENSE_RANK() OVER (ORDER BY SAL DESC) AS RNK
  FROM EMP
  WHERE rnk <= 5   => ERROR

  SELECT *
  FROM (SELECT empno,ename,sal,
             DENSE_RANK() OVER (ORDER BY SAL DESC) AS RNK
         FROM EMP) E
  WHERE rnk <= 5
```

=> to display top 5 max salaries

```
  SELECT DISTINCT sal
  FROM (SELECT empno,ename,sal,
             DENSE_RANK() OVER (ORDER BY SAL DESC) AS RNK
         FROM EMP) E
  WHERE rnk <= 5
  ORDER BY sal DESC
```

Example 3 :-

=> display first 5 rows from emp table ?

```
   SELECT *
   FROM (SELECT empno,ename,sal,deptno ,
                 row_number() over (order by empno asc) as rno
         FROM emp) E
   WHERE rno<=5
```

=> display 5th row ?

```
   SELECT *
   FROM (SELECT empno,ename,sal,deptno ,
                 row_number() over (order by empno asc) as rno
         FROM emp) E
   WHERE rno=5
```

=> display 5th record to 10th record ?

```
   SELECT *
   FROM (SELECT empno,ename,sal,deptno ,
                 row_number() over (order by empno asc) as rno
         FROM emp) E
   WHERE rno BETWEEN 5 AND 10
```

=> display even no rows ?

```
   SELECT *
   FROM (SELECT empno,ename,sal,deptno ,
                 row_number() over (order by empno asc) as rno
         FROM emp) E
   WHERE rno%2=0
```

06-sep-21

=> delete first 5 rows from emp table ?

```
   DELETE
   FROM (SELECT empno,ename,sal,deptno ,
                 row_number() over (order by empno asc) as rno
         FROM emp) E
   WHERE rno<=5    => ERROR
```

CTE :-

=> CTE stands for common table expression which is a named result which we can refer in another query like SELECT/INSERT/UPDATE/DELETE.

=> CTEs are used to simplify complex processing

=> using CTEs we can use result of one operation in another operation.

syntax :-

```
  WITH <name>
   AS
     (SELECT STATEMENT) ,
    <name>
      AS
         (SELECT STATEMENT)

   SELECT/INSERT/UPDATE/DELET STATEMENT
```

Example 1 :- delete first 5 rows from emp table ?

```
    WITH E
     AS
       (SELECT empno,ename,sal,deptno ,
                 row_number() over (order by empno asc) as rno
         FROM emp)
     DELETE FROM E WHERE rno<=5
```

Example 2 :- delete duplicate rows ?

```
  EMP33
  ENO   ENAME   SAL
  1     A       5000
  2     B       6000
  1     A       5000  => duplicate
  2     B       6000  => duplicate
  3     C       7000
```

```
SELECT eno,ename,sal,
    ROW_NUMBER() OVER (PARTITION BY eno,ename,sal ORDER BY eno ASC) as rno
FROM emp33

1  A  5000  1
1  A  5000  2

2  B  6000  1
```

```
2  B  6000  2

3  C  7000  1
```

=> to delete duplicates delete the records whose rno > 1 ?

```
 WITH E
 AS
  (SELECT eno,ename,sal,
    ROW_NUMBER() OVER (PARTITION BY eno,ename,sal ORDER BY eno ASC) as rno
  FROM emp33)
 DELETE FROM E WHERE RNO>1
```

method 2 :-

```
 EMP33
  ENO   ENAME   SAL
  1     A       5000
  2     B       6000
  1     A       5000  => duplicate
  2     B       6000  => duplicate
  3     C       7000
```

step 1 : - create temp table and copy distinct rows to temp table

```
 SELECT DISTINCT * INTO temp FROM emp33

 temp
 1      A       5000
 2      B       6000
 3      C       7000
```

step 2 :- truncate original table

```
   TRUNCATE TABLE emp33
```

step 3 :- copy data from temp to emp33

```
  INSERT INTO emp33
  SELECT * FROM temp
```

Assignment :-

  increment top 5 employee salaries by 10%

07-SEP-21

scalar subqueries :-

=> subqueries in SELECT clause are called scalar subqueries

```
  syn :- SELECT (subquery1),(subquery2),(subquery3)----- FROM tabname
```

 => subquery output acts like a column

Example 1 :-

```
  SELECT (SELECT COUNT(*) FROM emp) as emp,
         (SELECT COUNT(*) FROM dept) as dept

  emp     dept
  14      4
```

Example 2 :-

=> display dept wise total sal ?

```
   SELECT deptno,SUM(sal) as dept_totsal
   FROM emp
   GROUP BY deptno

 deptno         dept_totsal
   10           8750.00
   20           7100.00
   30           5300.00
```

=> display DEPTNO DEPT_TOTSAL   TOTSAL ?

```
   SELECT deptno,SUM(sal) as dept_totsal,
             (SELECT SUM(sal) FROM emp) as totsal
   FROM emp
   GROUP BY deptno

   10   8750.00 21150.00
   20   7100.00 21150.00
   30   5300.00 21150.00
```

=> display DEPTNO   DEPT_TOTSAL   TOTSAL   PCT ?

```
      PCT = (dept_totsal/totsal)*100


   SELECT deptno,SUM(sal) as dept_totsal,
             (SELECT SUM(sal) FROM emp) as totsal,
            (SUM(sal)/(SELECT SUM(sal) FROM emp))*100 as pct
   FROM emp
   GROUP BY deptno

   OR

   SELECT deptno,dept_totsal,totsal,(dept_totsal/totsal)*100 as pct
   FROM (SELECT deptno,SUM(sal) as dept_totsal,
                 (SELECT SUM(sal) FROM emp) as totsal
         FROM emp
         GROUP BY deptno) AS E
```

select
where clause
order by

functions
group by
joins
set operators
subqueries

08-sep-21

Database Transactions :-

=> a transaction is a unit of work that contains one or more dmls and must be saved as a whole or must be cancelled as a whole.

ex :- money transfer

```
      acct1-----------------1000------------------->acct2

      update1                                        update2
      (bal=bal-1000)                                 (bal=bal+1000)

      successful                                     failed      invalid

      failed                                         successful  invalid

      successful                                     successful   valid

      failed                                         failed       valid
```

=> every db transaction must gurantee a property called atomocity i.e. all or none
if transaction contains multiple operations , if all are successful then it must be saved , if one of the operation fails then entire transaction must be cancelled.

=> the following commands provided by sql server called TCL (transaction control language) commands to handle transactions

```
 1  COMMIT           => to save transaction
 2  ROLLBACK         => to cancel transaction
 3  SAVE TRANSACTION => to cancle transaction upto some point
```

=> every transaction has a begin point and an end point.

=> in sql server a txn begins implicitly with DML/DDL commands and ends implicitly with COMMIT.

=> we can also start transaction explicitly with "BEGIN TRANSACTION" command and ends explicitly with COMMIT/ROLLBACK.

Example 1 :-

```
 BEGIN TRANSACTION  => txn begins T1
 update1
 insert1
 update2
 insert2
 COMMIT           => txn ends
```

=> if txn ends with commit then it is called successful transaction and operations are saved

Example 2 :-

```
 BEGIN TRANSACTION  => txn begins T1
 update1
 insert1
 update2
 insert2
 ROLLBACK    => txn ends
```

=> if txn ends with rollbck then it is called aborted transaction and operations are cancelled

Example 3 :-

```
 create table a(a int)
 begin transaction
 insert into a values(10)
 insert into a values(20)
 insert into a values(30)
 insert into a values(40)
 rollback

 output :-

 create table a  => saved
 inserts         => cancelled
```

Example 4 :-

```
 create table a(a int)
 begin transaction
 insert into a values(10)
 insert into a values(20)
 commit
 insert into a values(30)
 insert into a values(40)
 rollback

output :- create table a => saved
          insert 10,20   => saved (commit)
          insert 30,40   => saved (implicit commit)
```

Example 5 :-

```
 create table a(a int)
 begin transaction
 insert into a values(10)
 insert into a values(20)
 rollback
 insert into a values(30)
 insert into a values(40)
 rollback
```

output :-

```
 create table  => saved
 insert 10,20  => cancelled
 insert 30,40  => saved
```

SAVE TRANSACTION :-

=> we can declare save transaction and we can rollback upto the save transaction.

=> using save transaction we can cancel part of the transaction.

```
 create table a(a int)
 begin transaction
 insert into a values(10)
 insert into a values(20)
 SAVE TRANSACTION ST1
 insert into a values(30)
 insert into a values(40)
 SAVE TRANSACTION ST2
 insert into a values(50)
 insert into a values(60)
 ROLLBACK TRANSACTION ST1
 COMMIT

 SELECT * FROM a

 10
 20
```

11-SEP-21

Database Security :-

```
 1 logins     => provides security at server level
 2 users      => provides security at db level
 3 privileges => provides security at table level
 4 views      => provides security at row & col level
```

13-sep-21

DB objects / SCHEMA objects :-

TABLES

VIEWS
SYNONYMS
SEQUENCES
INDEXES

VIEWS :-

=> a view is a subset of a table
=> a view is a virtual table
=> a view is a representation of a query
=> views are created

1 to provide security
2 to reduce complexity

=> view provides another level of security by granting specific rows & columns to users

=> views are 2 types

1 simple views
2 complex views

1 simple views :-

=> a view is called simple if it is based on single table

```
CREATE VIEW <NAME>
AS
SELECT STATEMENT
```

Example :-

```
CREATE VIEW V1
AS
SELECT empno,ename,job,deptno
FROM emp
```

=> when the above command is executed then sql server creates view v1 and stores query but not query output (data).

=> a view is called virtual because it doesn't store data and doesn't occupy memory and it always derives data from base table

```
SELECT * FROM V1
```

=> when above query submitted to sql sever it executes the query as follows

```
SELECT * FROM (SELECT empno,ename,job,deptno FROM emp)
```

Granting permissions on view to user :-

```
 GRANT SELECT,INSERT,UPDATE,DELETE ON V1 TO VIJAY

=> after granting permission on view to vijay , vijay can access emp table
through view

 VIJAY :-
 ---------

1 SELECT * FROM V1

2 INSERT INTO V1 VALUES(6666,'ABC','CLERK',20)

=> above insert command inserts row into emp table and fields which are not
included
   in view are filled with nulls.

3 UPDATE V1 SET JOB='MANAGER' WHERE EMPNO=6666

4 UPDATE V1 SET SAL=3000 WHERE EMPNO=6666 => ERROR

ROW LEVEL SECURITY :-
---------------------

CREATE VIEW V2
AS
SELECT empno,ename,job,deptno
FROM emp
WHERE deptno=20

GRANT SELECT,INSERT,UPDATE,DELETE ON V2 TO VIJAY

VIJAY :-

INSERT INTO V2 VALUES(5555,'XYZ','CLERK',30) => 1 ROW AFFECTED

=> above insert command executed successfully even though it is violating
where condition

WITH CHECK OPTION :-
--------------------

=> if view created with "WITH CHECK OPTION" then any dml command through view
violates
   where condition that dml is not accepted.

CREATE VIEW V3
AS
SELECT empno,ename,job,deptno
FROM emp
WHERE deptno=20
WITH CHECK OPTION

GRANT SELECT,INSERT,UPDATE,DELETE ON V3 TO VIJAY

VIJAY :-

INSERT INTO V3 VALUES(4444,'KLM','CLERK',30) => ERROR
```

14-sep-21

complex views :-

=> a view said to be complex view

```
  1 if it is based on multiple tables
  2 if query contains group by clause
                      having clause
                      distinct clause
                      aggregate functions
                      subqueries
                      set operators
```

=> with the help of views complex queries can be converted into simple queries

Example 1 :-

```
 CREATE VIEW CV1
 AS
 SELECT e.empno,e.ename,e.sal,
        d.deptno,d.dname,d.loc
 FROM emp e INNER JOIN dept d
   ON e.deptno = d.deptno
```

=> after creating view , whenever we want emp & dept details instead of writing join query write the simple query as follows

```
     SELECT * FROM CV1
```

Example 2 :-

```
 CREATE VIEW CV2
 AS
 SELECT d.dname,MIN(e.sal) as minsal,
                MAX(e.sal) as maxsal,
                SUM(e.sal) as totsal,
                COUNT(e.empno) as cnt
 FROM emp e INNER JOIN dept d
   ON e.deptno = d.deptno
 GROUP BY d.dname
```

=> after creating view whenever we want dept wise summary then execute the following query

```
  select * from cv2
```

=> diff b/w simple and complex views ?

	simple	complex
1	based on single table	based on multiple tables

2	query performs simple operations	query performs complex operations like joins group by etc
3	always updatable i.e. allows dmls	not updatable i.e. doesn't allow dmls

Question :-

```
CREATE VIEW V10
AS
SELECT * FROM EMP

ALTER TABLE EMP
   ADD GENDER CHAR(1)
```

=> is this new column is added to view or not ?

ans :- by default column is not added , to add this column to view it must be recreated

```
CREATE OR ALTER VIEW V10
AS
SELECT * FROM EMP
```

=> display list of views created by user ?

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS
```

Droping view :-

```
DROP VIEW V1
```

=> if we drop table what about views created on table ?

ANS :- not dropped but views cannot be queried

WITH SCHEMABINDING :-

=> if view created with "WITH SCHEMABINDING" then sql server will not allow users to drop table if any view exists on the table , To drop table first we need to drop view.

Rules ;-

1 "*" is not allowed in SELECT clause
2 tablename should be prefixed with schema name

```
CREATE VIEW V20
WITH SCHEMABINDING
AS
```

```
 SELECT DEPTNO,DNAME,LOC FROM DBO.DEPT

 DROP TABLE DEPT  => ERROR
```

15-SEP-21

SEQUENCES :-

=> sequences are created to generate sequence numbers
=> sequences are created to auto increment column values

syn :-

```
 CREATE SEQUENCE <NAME>
 START WITH <value>
 INCREMENT BY <value>
 MAXVALUE <value>
 MINVALUE <value>
 CYCLE/NOCYCLE
```

Example:-

```
 CREATE SEQUENCE S1
 START WITH 1
 INCREMENT BY 1
 MAXVALUE 5


 CREATE TABLE student
 (
   sid  int,
   sname varchar(10)
 )

INSERT INTO student values(next value for s1,'A')
INSERT INTO student values(next value for s1,'B')
INSERT INTO student values(next value for s1,'C')
INSERT INTO student values(next value for s1,'D')
INSERT INTO student values(next value for s1,'E')
INSERT INTO student values(next value for s1,'F')  => ERROR
```

calling sequence in update command :-

```
CREATE SEQUENCE S2
START WITH 100
INCREMENT BY 1
MAXVALUE 1000
```

=> use above sequence update empno ?

```
  UPDATE emp SET empno = next value for s2
```

calling sequence in expressions :-

```
CREATE TABLE INVOICE
(
  INVNO  VARCHAR(30),
  INV_DT DATETIME
)



CREATE SEQUENCE S3
START WITH 1
INCREMENT BY 1
MAXVALUE 1000

INSERT INTO INVOICE VALUES('KLM/' + FORMAT(getdate(),'MMdd') + '/' +
CAST(NEXT VALUE FOR S3 AS VARCHAR),getdate())
INSERT INTO INVOICE VALUES('KLM/' + FORMAT(getdate(),'MMdd') + '/' +
CAST(NEXT VALUE FOR S3 AS VARCHAR),getdate())
INSERT INTO INVOICE VALUES('KLM/' + FORMAT(getdate(),'MMdd') + '/' +
CAST(NEXT VALUE FOR S3 AS VARCHAR),getdate())

SELECT * FROM invoice
```

```
INVNO          INV_DT
KLM/0915/1
KLM/0915/2
```

CYCLE / NOCYCLE :-

=> default is NOCYCLE , if sequence created with NOCYCLE then it starts from start with and generates upto max and after reaching max then it stops.

=> if sequence created with CYCLE then it starts from start with and generates upto to max and after reaching max then it will be reset to min.

```
  CREATE SEQUENCE S4
  START WITH 1
  INCRMENT BY 1
  MAXVALUE 5
  MINVALUE 1
  CYCLE

 INSERT INTO STUDENT VALUES(NEXT VALUE FOR S4,'A')
 --------------------------------
 -------------------------------
 -----------------------------

 SELECT * FROM STUDENT
```

```
1       A
2       B
3       C
4       D
5       E
1       F
```

```
2       G
3       H
4       I
5       J
```

How to reset sequence manually :-

```
 CREATE SEQUENCE S5
 START WITH 1
 INCREMENT BY 1
 MAXVALUE 1000
```

=> after reaching 50 reset to 1 ?

```
   ALTER SEQUENCE S5 RESTART WITH 1
```

How to alter sequence :-

```
 ALTER SEQUENCE S2 MAXVALUE 5000
```

=> display list of sequences created by user ?

```
  SELECT * FROM INFORMATION_SCHEMA.SEQUENCES
```

Droping sequence :-

```
  DROP SEQUENCE S1
```

16-sep-21

INDEXES :-

=> index is also a db object created to improve performance of data accessing.

=> index in db is similar to index is textbook , in textbook using index a particular topic can be located fastly and in db using index a particular record can be located fastly.

=> when user submits a query , sql server goes through following steps to execute the query

1 parsing
2 optimization
3 execution

parsing :-

1 checks syntax
2 checks semantics
checks table exists in the db or not

checks columns belongs to the table or not
user has permission to access table or not

optimization :-

=> sql server prepares plans to execute the query
=> sql server prepares mainly two plans

1 table scan
2 index scan

=> estimate the cost of each plan and selects best plan i.e. plan that takes less cost
=> in table scan sql server scans whole table
=> in index scan on avg sql server scans half of the table

execution :-

=> sql server executes the query according to plan selected by optimizer.

=> indexes are created on columns and that column is called index key

=> indexes are created on columns

1 which are frequently accessed in where clause
2 which are used in join operation

Types of indexes :-
----

1 Non clustered Index
    simple
    composite
    unique
2 Clustered Index

Simple Non Clustered Index :-

=> if index created on single column then index is called simple index.

syn :- CREATE INDEX <NAME> ON <TABNAME>(COLNAME)

Ex :- CREATE INDEX I1 ON EMP(SAL)

```
 EMP                                                3000
 SAL
 5000
 1000                                   2000                        4000
 3000
 2000                         1000 *          2500 *          4000 *
5000 *
 4000                         1500 *          3000 *,*
 1500                         2000 *
 3000
 2500
```

=> SQL SERVER uses above index for the queries where condition based on sal column

```
        SELECT * FROM emp WHERE sal=3000 ;
        SELECT * FROM emp WHERE sal>=3000 ;
        SELECT * FROM emp WHERE sal<=3000 ;
```

17-sep-21

composite index :-

=> if index created on multiple columns then index is called composite index

```
      CREATE INDEX I2 ON EMP(DEPTNO,JOB)
```

=> sql server uses above index when where condition based on leading column of the index i.e. deptno

```
   SELECT * FROM emp WHERE deptno=20;                 (index)
   SELECT * FROM emp WHERE deptno=20 and job='clerk'    (index)
   SELECT * FROM emp WHERE job='clerk'                (table)
```

unique index :-

=> unqiue index doesn't allow duplicate values into the column on which index is created

```
  ex :- CREATE UNIQUE INDEX I3 ON EMP(ENAME)
```

```
                                    K

                     G                            Q

        ADAMS *          JAMES *        MARTIN *        SCOTT *
         ALLEN *           JONES *         MILLER *        SMITH *
         BLAKE *           KING *
```

```
    INSERT INTO emp(empno,ename,sal) VALUES(100,'BLAKE',5000)  => ERROR
because unique index
                                                                  doesn't
allow duplicates
```

=> what are the different methods to enforce unqiueness ?

1 declare primary key / unique constraint
2 create unique index

=> primary key / unique columns are implicitly indexed by sql server , sql sever creates a unique

index on primary key / unique columns and unique index doesn't allow duplicates so primary key/
unique also doesn't allow duplicates .

CLUSTERED INDEX :-

=> a non clustered index stores pointers to actual records where as clustered index stores actual records

=> in non clustered index order of the records in index and order of the records in table is not same
where as in clustered index this order will be same.

ex :-

```
CREATE TABLE cust
(
   cid  int,
   cname varchar(10)
)

CREATE CLUSTERED INDEX I5 ON CUST(cid)

INSERT INTO CUST VALUES(10,'A')
INSERT INTO CUST VALUES(80,'B')
INSERT INTO CUST VALUES(40,'C')
INSERT INTO CUST VALUES(60,'D')
INSERT INTO CUST VALUES(20,'E')
```

```
                         50

                30                 70

        10  A       40  C    60  D      80  B
         20  E
```

SELECT * FROM CUST => sql server goes to clustered index and reads nodes from left to right

```
10 A
20 E
40 C
60 D
80 B
```

=> only one clustered index is allowed per table and sql server creates clustered index on priamry key column

diff b/w non clustered and clustered index ?

	non clustered	clustered
1	stores pointers to actual records	stores actual records

```
    2     order of the records in table                 order of the records
in tablea and index
          and index will not be same                    will be same

    3     requires extra storage                        doesn't need extra
storage

    4     requires two lookups                          requires single
lookup

    5     sql server 999 non clustered                  only one clustered
index is allowed per table
          indexes per table

    6     created explicitly                            created implicitly
on primary key column
```

=> display list of indexes ?

```
sp_helpindex emp
```

Droping index :-

```
DROP INDEX I1
```

=> if we drop table what about indexes created on table ?

ans :- indexes are also dropped

18-sep-21

synonyms :-

=> a synonym is another name or alternative name to table or view.

=> when tablename is lengthy or complex then we can give a simple and short name to the table and instead of using tablename we can use that simple and short name called synonym.

syn :- CREATE SYNONYM <NAME> FOR <TABNAME>

ex :- CREATE SYNONYM E FOR EMP

=> after creating synonym instead of using tablename we can use synonym name in SELECT/INSERT/UPDATE/DELETE queries

```
1  SELECT * FROM E

2  UPDATE E SET COMM=500 WHERE EMPNO=7499
```

MERGE command :-

=> merge command is used to merge data into a table.
=> merge command is the combination of insert,update,delete
=> used to apply changes made to one table to another table
=> used to manage replicas (duplicate copy)

scenario :-

17th sep

CUSTS

CID	CNAME	ADDR
1	A	HYD
2	B	MUM

=> create replica for custs ?

```
  SELECT * INTO CUSTT FROM CUSTS
```

CUSTT

CID	CNAME	ADDR
1	A	HYD
2	B	MUM

18th sep

CUSTS

CID	CNAME	ADDR
1	A	BLR => updated
2	B	MUM
3	C	DEL => inserted

=> whatever changes (insert,update) made to custs we need to apply these changes to custt ,
instead of executing insert & update seperately we can combine these two commands into
one command called merge

```
syntax :- MERGE INTO <TARGET-TABLE> <ALIAS>
          USING <SOURCE-TABLE> <ALIAS>
          ON (COND)
          WHEN MATCHED THEN
            UPDATE
          WHEN NOT MATCHED BY TARGET THEN
            INSERT
          WHEN NOT MATCHED BY SOURCE THEN
            DELETE
```

Example 1 :- replicating inserts,updates

```
 MERGE INTO CUSTT T
 USING CUSTS S
 ON (S.CID=T.CID)
 WHEN MATCHED THEN
  UPDATE SET T.CADDR = S.CADDR
 WHEN NOT MATCHED THEN
   INSERT VALUES(S.CID,S.CNAME,S.CADDR);
```

20-sep-21

Example 2 :- replicating inserts,updates,deletes

```
 CUSTS
 CID    CNAME    ADDR
 1      A        BLR  => updated
 2      B        MUM
 3      C        DEL => delete
 4      D        CHE => inserted

 CUSTT
 CID    CNAME    ADDR
 1      A        HYD
 2      B        MUM
 3      C        DEL
```

```
MERGE INTO CUSTT T
  USING CUSTS S
  ON (S.CID=T.CID)
  WHEN MATCHED THEN
   UPDATE SET T.CADDR = S.CADDR
  WHEN NOT MATCHED THEN
    INSERT VALUES(S.CID,S.CNAME,S.CADDR)
  WHEN NOT MATCHED BY SOURCE THEN
    DELETE
```

PIVOT operator :-

=> used for cross tabulation or matrix report
=> used for converting rows into columns

```
 SELECT columns
 FROM (SELECT required data) AS <ALIAS>
 PIVOT
  (
    aggr-expr FOR colname IN (V1,V2,V3,--)
  ) AS <PIVOT-TABLE-NAME>
 ORDER BY col ASC/DESC
```

Example 1 :-

```
                       10             20             30

        ANALYST        ??             ???            ???

        CLERK          ??             ???            ???

        MANAGER        ???            ???            ???

        SALESMAN       ???            ???            ???
```

```
SELECT *
FROM (SELECT deptno,job,sal FROM emp) AS E
PIVOT
(
 SUM(sal) FOR deptno IN ([10],[20],[30])
) AS PIVOT_TBL
ORDER BY job ASC
```

Example 2 :-

```
                1      2      3       4

       1980     ?      ?       ?       ?

       1981     ?      ?       ?       ?

       1982     ?      ?       ?       ?

       1983     ?      ?       ?       ?
```

```
SELECT *
FROM (SELECT DATEPART(yy,hiredate) as year,
             DATEPART(qq,hiredate) as qrt,
             empno
      FROM emp) AS E
PIVOT
 (
   COUNT(empno) FOR  qrt IN ([1],[2],[3],[4])
 ) AS PIVOT_TBL
ORDER BY year ASC
```

Example 3 :- converting rows into columns

STUDENT

SNO	SNAME	SUBJECT	MARKS
1	A	MAT	90
1	A	PHY	80
1	A	CHE	60
2	B	MAT	50
2	B	PHY	40
2	B	CHE	60

OUTPUT :-

```
SNO    SNAME   MAT     PHY     CHE
 1     A       90      80      60
 2     B       50      40      60
```

```
SELECT *
FROM STUDENT
PIVOT
 (
  SUM(MARKS) FOR SUBJECT IN ([MAT],[PHY],[CHE])
 ) AS PIVOT_TBL
ORDER BY SNO ASC
```

21-sep-21 TSQL programming

Features :-
----

1 improves performance :-

=> using TSQL programming we can group sql commands into one program and we submit that program to sql server , so in TSQL programming no of requests and response between user and sql server are reduced and performance is improved.

2 supports conditional statements :-

=> TSQL supports conditional statements like if-then-else and using this we can execute commands based on conditions

3 supports loop :-

=> loops are used to execute statements repeatedly multiple times

4 supports error handling :-

=> in tsql , if any statement causes runtime error then sql server displays error message , using error handling we can replace system generated message with our own simple and user friendly message

5 supports reusability :-

=> TSQL programs can be centralized i.e. stored in db and applications which are connected to db can reuse these programs.

6 supports security :-

=> because these programs are stored in db and only authorized users can execute these programs but not unauthorized users.

=> TSQL blocks are 2 types

```
 1 anonymous blocks
 2 named blocks
     stored procedures
     stored functions
     triggers
```

Anonymous Blocks :-

=> TSQL blocks without name are called anonymous blocks
=> the following statments are used in TSQL programming

```
 1 DECLARE
 2 SET
 3 PRINT
```

DECLARE statement :-

=> used to declare variables

```
   syn :- DECLARE @var datatype(size)

   ex :-  DECLARE @x int
          DECLARE @s varchar(10)
          DECLARE @d date

          DECLARE @x int,@s varchar(10),@d date
```

SET statement :-

=> used to assign value to variable

```
   SET @var = value

   SET @x=100
   SET @s='abc'
   SET @d=GETDATE()
```

PRINT statement :-

=> used to print messages or variable values

```
    PRINT @x
    PRINT @s
    PRINT @d
```

=> write a prog to add two numbers ?

```
  DECLARE @a int,@b int,@c int
  SET @a=100
  SET @b=200
  SET @c=@a+@b
  PRINT @c
```

=> write a prog to input date and print day of the week ?

```
  DECLARE @d date
  SET @d='2021-10-15'
  PRINT DATENAME(dw,@d)
```

DB programming with TSQL :-

=> to perform operations over db execute SQL commands from tsql program
=> the following commands can be executed from tsql program

1 DML (insert,update,delete,merge)
2 DQL (select)
3 TCL (commit,rollback,save transaction)

SELECT stmt syntax :-

```
 SELECT @var1=col1,
        @var2=col2,
        @var3=col3,--------
 FROM tabname
 WHERE condition
```

22-sep-21

=> write a prog to input empno and print name & salary ?

```
  DECLARE @eno int,@name varchar(10),@sal money
  SET @eno=7844
  SELECT @name=ename,@sal=sal FROM emp WHERE empno=@eno
  PRINT   @name + '  earns  '  +  CAST(@sal AS varchar)
```

=> write a prog to input empno and calculate experience ?

```
    DECLARE @eno int,@hire date,@expr tinyint
    SET @eno=7369
    SELECT @hire=hiredate FROM emp WHERE empno=@eno
    SET @expr = DATEDIFF(yy,@hire,getdate())
    PRINT CAST(@expr as varchar) + ' years'
```

=> write a prog to input empno and calculate total sal ?

total sal = sal + comm

```
    DECLARE @eno int,@sal money,@comm money,@totsal money
    SET @eno=7566
    SELECT @sal=sal,@comm=comm FROM emp WHERE empno=@eno
    SET @totsal = @sal+ISNULL(@comm,0)
    PRINT 'Total Sal = ' + CAST(@totsal as varchar)
```

23-sep-21

conditional statements :-

```
 1 if-else
 2 multi if
 3 nested if

1 if-else :-
------------

 if cond
  begin
    statements
  end
 else
  begin
   statements
  end

2 multi if :-
-------------

 if cond1
 begin
   statements
 end
else if cond2
 begin
   statements
 end
else if cond3
 begin
   statements
 end
else
 begin
   statements
 end

3 nested if :-
---------------

 if cond
 begin
   if cond
    begin
       statements
    end
   else
    begin
      statements
    end
 end
else
  begin
    statements
 end

Example 1 :-
```

=> Write a prog to input empno and calculate experience , if expr > 40 the delete record from table ?

```
 DECLARE @eno int,@hire date,@expr int
 SET @eno=7369
 SELECT @hire=hiredate FROM emp WHERE empno=@eno
 SET @expr=DATEDIFF(yy,@hire,GETDATE())
 IF @expr>40
   DELETE FROM emp WHERE empno=@eno
```

=> write a prog to increment specific employee sal by specific amount and after increment if sal exceeds 5000 then cancel that increment ?

```
  DECLARE @eno int,@amt money,@sal money
  SET @eno=7788
  SET @amt=2500
  BEGIN TRANSACTION
  UPDATE emp SET sal=sal+@amt WHERE empno=@eno
  SELECT @sal=sal FROM emp WHERE empno = @eno
  IF @sal>5000
      ROLLBACK
  ELSE
     COMMIT
```

=> write a prog to input empno and increment employee salary as follows ?

```
   if job=CLERK incr sal by 10%
          SALESMAN          15%
          MANAGER           20%
          others            5%
```

```
 DECLARE @eno int,@job varchar(10),@pct
 SET @eno-7844
 SELECT @job=job FROM emp WHERE empno=@eno
 IF @job='CLERK'
    SET @pct=10
 ELSE IF @job='SALESMAN'
    SET @pct=15
 ELSE IF @job='MANAGER'
   SET @pct=20
 ELSE
   SET @pct=5
 UPDATE emp SET sal=sal+(sal*@pct/100) WHERE empno=@eno
```

24-sep-21

ACCOUNTS

ACCNO	NAME	BAL
100	A	10000
101	B	20000

=> write a prog to process bank transaction (w/d) ?

```
 DECLARE @acno int,@type char(1),@amt money,@bal money
 SET @acno=100
 SET @type='W'
```

```
SET @amt=1000
IF @type='W'
 BEGIN
   SELECT @bal=bal FROM accounts WHERE accno=@acno
   IF @amt > @bal
     PRINT 'insufficient balance'
   ELSE
     UPDATE accounts SET bal=bal-@amt WHERE accno=@acno
 END
ELSE IF @type='D'
    UPDATE accounts SET bal=bal+@amt WHERE accno=@acno
ELSE
   PRINT 'invalid transaction'
```

=> write a prog for money transfer ?

=> write a prog to calculate particular student total,avg,result and insert into result table ?

STUDENT

SNO	SNAME	S1	S2	S3
1	A	80	90	70
2	B	50	60	30

RESULT

SNO	STOT	SAVG	SRES

```
DECLARE @sno int,@s1 int,@s2 int,@s3 int,@total int,@avg decimal(5,2),@res char(4)
SET @sno=1
SELECT @s1=s1,@s2=s2,@s3=s3 FROM student WHERE sno=@sno
SET @total=@s1+@s2+@s3
SET @avg=@total/3
IF @s1>=35 AND @s2>=35 AND @s3>=35
 SET @res='pass'
ELSE
 SET  @res='fail'
INSERT INTO result VALUES(@sno,@total,@avg,@res)
```

WHILE LOOP :-

=> loops are used to execute statements repeatedly multiple times

```
WHILE(cond)
BEGIN
  statements
END
```

if cond=true loop continues
if cond=false loop terminates

27-sep-21

=> write a prog to print nos from 1 to 20 ?

```
DECLARE @x tinyint=1
WHILE(@x<=20)
BEGIN
   PRINT @x
   SET @x = @x+1
END
```

=> write a prog to print 2022 calendar ?

```
2022-01-01     ?
2022-01-02     ?

2022-12-31     ?
```

```
DECLARE @d1 date,@d2 date
SET @d1 = '2022-01-01'
SET @d2 = '2022-12-31'
WHILE(@d1<=@d2)
BEGIN
  PRINT CAST(@d1 AS VARCHAR) +  '    '  + DATENAME(dw,@d1)
  SET @d1 = DATEADD(dd,1,@d1)
END
```

=> write a prog to sundays between two given dates ?

```
DECLARE @d1 date,@d2 date
SET @d1 = '2022-01-01'
SET @d2 = '2022-12-31'
WHILE(@d1<=@d2)
BEGIN
  IF DATENAME(dw,@d1)='sunday'
     PRINT CAST(@d1 AS VARCHAR) +  '    '  + DATENAME(dw,@d1)
  SET @d1 = DATEADD(dd,1,@d1)
END
```

=> prog to print tables upto 5 ?

```
  DECLARE @i int=2,@j int
  WHILE(@i<=5)
  BEGIN
     SET @j=1
     WHILE(@j<=10)
     BEGIN
       PRINT CAST(@i AS VARCHAR) + 'X' + CAST(@j AS VARCHAR) + ' = ' +
CAST(@i*@j AS VARCHAR)
       SET @j=@j+1
     END
     SET @i = @i+1
  END
```

=> write a prog to input string and print it in following pattern ?

input :- BHARAT

```
  output :-
 B
 H
 A
 R
 A
 T
```

```
 DECLARE @s varchar(10),@x int=1
 SET @s='BHARAT'
 WHILE(@x<=LEN(@s))
 BEGIN
   PRINT SUBSTRING(@s,@x,1)
   SET @x=@x+1
END
```

=> write a prog to input string and print in following pattern ?

```
 INPUT :- INDIA
 OUTPUT :-

 I
 IN
 IND
 INDI
 INDIA
```

```
 DECLARE @s varchar(10),@x int=1
 SET @s='INDIA'
 WHILE(@x<=LEN(@s))
 BEGIN
   PRINT SUBSTRING(@s,1,@x)
   SET @x=@x+1
 END
```

28-sep-21

CURSORS :-

=> CURSORS are used to access row-by-row into tsql program.

=> from tsql program if we submit a query to sql server , it goes to db and and fetch the data
and copies that data into temporary memory and using cursor we can give name to that memory
and access row-by-row into tsql program and process the row.

=> follow below steps to use cursor

1 declare cursor
2 open cursor
3 fetch records from cursor
4 close cursor
5 deallocate cursor

declaring cursor :-

```
SYN :-   DECLARE <CURSOR-NAME> CURSOR FOR SELECT STATEMENT

EX  :-  DECLARE C1 CURSOR FOR SELECT * FROM emp
```

opening cursor :-

```
   open <cursor-name>

   OPEN C1
```

=> select statement submitted to sql server
=> data returned by query is copied to temporary storage
=> cursor c1 points to temporary storage

fetching records from cursor :-

=> "FETCH" statement is used to fetch record from cursor to program

```
   FETCH NEXT FROM <CURSOR-NAME> INTO VARIABLES ;

   FETCH NEXT FROM C1 INTO @a,@b,@c
```

=> a fetch statement fetches one row at a time but to process multiple rows fetch stmt should be executed multiple times so fetch stmt should be inside a loop.

closing cursor :-

```
   close c1;
```

deallocating cursor :-

```
  deallocate c1
```

@@FETCH_STATUS :-

=> used to find whether fetch is successful or not

```
   0   => fetch successful
  -1   => fetch unsuccessful
```

=> write a prog to print all employee names and salaries ?

```
  DECLARE C1 CURSOR FOR SELECT ename,sal FROM emp
  DECLARE @name varchar(10),@sal money
  OPEN C1
  FETCH NEXT FROM C1 INTO @name,@sal
  WHILE(@@FETCH_STATUS=0)
```

```
   BEGIN
     PRINT @name + '   ' + CAST(@sal as varchar)
     FETCH NEXT FROM C1 INTO @name,@sal
   END
      CLOSE C1
      DEALLOCATE C1
```

=> write a prog to calculate total salary without using SUM function ?

```
   DECLARE C1 CURSOR FOR SELECT sal FROM emp
   DECLARE @sal MONEY,@total MONEY = 0
   OPEN C1
   FETCH NEXT FROM C1 INTO @sal
   WHILE(@@FETCH_STATUS=0)
   BEGIN
     SET @total = @total + @sal
     FETCH NEXT FROM C1 INTO @sal
   END
     PRINT @total
     CLOSE C1
     DEALLOCATE C1
```

=> write a prog to calculate max salary ?

=> write a prog to calculate min salary ?

=> write a prog to calculate all the students total,avg,result and insert into result table ?

STUDENT

SNO	SNAME	S1	S2	S3
1	A	80	90	70
2	B	50	60	30

RESULT

SNO	STOT	SAVG	SRES

```
 DECLARE C1 CURSOR FOR SELECT SNO,S1,S2,S3 FROM STUDENT
 DECLARE @sno int,@s1 int,@s2 int,@s3 int,@total int,@avg decimal(5,2),@res
char(4)
 OPEN C1
 FETCH NEXT FROM C1 INTO @sno,@s1,@s2,@s3
 WHILE(@@FETCH_STATUS=0)
 BEGIN
  SET @total=@s1+@s2+@s3
  SET @avg=@total/3
  IF @s1>=35 AND @s2>=35 AND @s3>=35
   SET @res='pass'
  ELSE
   SET  @res='fail'
  INSERT INTO result VALUES(@sno,@total,@avg,@res)
  FETCH NEXT FROM C1 INTO @sno,@s1,@s2,@s3
 END
   CLOSE C1
   DEALLOCATE C1
```

29-sep-21

raise_salary

empno	pct
7369	15
7499	10
7521	12
7566	20
7654	15

=> write a prog to increment employee salaries based on pct in raise_salary table ?

```
  DECLARE C1 CURSOR FOR SELECT * FROM raise_salary
  DECLARE @eno int,@pct int
  OPEN C1
  FETCH NEXT FROM C1 INTO @eno,@pct
  WHILE(@@FETCH_STATUS=0)
  BEGIN
    UPDATE emp SET sal=sal+(sal*@pct/100) WHERE empno=@eno
    FETCH NEXT FROM C1 INTO @eno,@pct
  END
     CLOSE C1
     DEALLOCATE C1
```

SCROLLABLE CURSOR :-

=> by default cursor is forward only cursor and forward only cursor supports only forward navigation
but doesn't support backward navigation.

=> if cursor declared with SCROLL then it is called scrollable cursor and a scorllable cursor supports
both forward and backward navigation.

=> forward only cursor supports only FETCH NEXT statement but SCROLLABLE cursor supports the following
fetch statements

```
   FETCH FIRST         => fetches first record
   FETCH NEXT          => fetches next record
   FETCH PRIOR         => fetches previous record
   FETCH LAST          => fetches last record
   FETCH ABSOLUTE N    => fetches Nth record from first record
   FETCH RELATIVE N     => fetches Nth record from current record
```

Example 1 :-

```
DECLARE C1 CURSOR SCROLL FOR SELECT ename FROM emp
DECLARE @name VARCHAR(10)
OPEN C1
FETCH FIRST FROM C1 INTO @name
PRINT @name
FETCH ABSOLUTE 5 FROM C1 INTO @name
PRINT @name
FETCH RELATIVE 5 FROM C1 INTO @name
PRINT @name
```

```
 FETCH LAST FROM C1 INTO @name
 PRINT @name
 FETCH PRIOR FROM C1 INTO @name
 PRINT @name
 CLOSE C1
 DEALLOCATE C1
```

Example 2 :- write a prog to print evern 5th record ?

```
  DECLARE C1 CURSOR SCROLL FOR SELECT ename FROM emp
  DECLARE @name VARCHAR(10)
  OPEN C1
  FETCH RELATIVE 5 FROM C1 INTO @name
  WHILE(@@FETCH_STATUS=0)
  BEGIN
   PRINT @name
   FETCH RELATIVE 5 FROM C1 INTO @name
 END
   CLOSE C1
   DEALLOCATE C1
```

Example 3 :- write a prog to print names last to first ?

30-sep-21

ERROR HANDLING / EXCEPTION HANDLING :-

1 syntax errors
2 logical errors
3 runtime errors

=> errors that are raised during program execution are called runtime errors

```
 ex :- declare @x tinyint

       set @x=1000    => runtime error
```

=> in TSQL programming if any statement causes runtime error then sql server displays error message.
   To replace system generated message with our own simple and user friendly message then
   we need to handle that runtime error

=> to handle runtime error include a block TRY---CATCH block

```
   BEGIN TRY
    statemens    => causes runtime error
  END TRY
  BEGIN CATCH
    statements   => handles runtime error
  END CATCH
```

=> in try block if any statement causes runtime error then control will be transferred to catch block
and executes statements in catch block.

Example 1 :-

```
  declare @a tinyint,@b tinyint,@c tinyint
  begin try
  set @a=100
  set @b=0
  set @c=@a/@b
  print @c
  end try
  begin catch
   print 'ERROR'
  end catch
```

ERROR HANDLING FUNCTIONS :-

```
1 ERROR_NUMBER    => returns error number
2 ERROR_MESSAGE   => returns error message
3 ERROR_SEVERITY  => returns error severity level
4 ERROR_STATE     => returns error state
5 ERROR_LINE      => returns line number
```

Example 2 :-

```
  declare @a tinyint,@b tinyint,@c tinyint
  begin try
  set @a=100
  set @b=0
  set @c=@a/@b
  print @c
  end try
  begin catch
    if error_number()=220
       print 'value exceeding limit'
    else if error_number()=8134
       print 'divisor cannot be zero'
  end catch
```

Example 3 :-

```
 CREATE TABLE emp44
 (
   empno int PRIMARY KEY,
   ename VARCHAR(10) NOT NULL,
   sal MONEY CHECK(sal>=3000)
 )
```

=> write a prog to insert record into above table

```
 DECLARE @eno int,@name varchar(10),@sal money
 BEGIN TRY
 SET @eno=100
 SET @name='abc'
 SET @sal=4000
 INSERT INTO emp44 VALUES(@eno,@name,@sal)
 END TRY
 BEGIN CATCH
   IF ERROR_NUMBER()=2627
     PRINT 'empno should not be duplicate'
   ELSE IF ERROR_NUMBER()=515
     PRINT 'name should not be null'
   ELSE IF ERROR_NUMBER()=547
     PRINT 'sal >=3000'
 END CATCH
```

USER DEFINE ERRORS :-

=> errors raised by user are called user defined errors
=> user define errors are raised by using by using

```
   RAISERROR(error msg,error severity level,error state)
```

Example 1 :-

=> write a prog to increment specific employee sal by specific amount but sunday updates are not allowed

```
 DECLARE @eno int,@amt money
 SET @eno=7788
 SET @amt=1000
 IF DATENAME(dw,GETDATE())='SUNDAY'
    RAISERROR('sunday not allowed',15,1)
 ELSE
   UPDATE emp SET sal=sal+@amt WHERE empno=@eno
```

Example 2 :-

```
 CREATE TABLE emp44
(
  empno int primary key,
  ename varchar(10) not null,
  sal money check(sal>=3000)
 )

 DECLARE @eno int,@name varchar(10),@sal money
 DECLARE @msg VARCHAR(100)
 BEGIN TRY
 SET @eno=100
 SET @name='abc'
 SET @sal=4000
 INSERT INTO emp44 VALUES(@eno,@name,@sal)
 END TRY
 BEGIN CATCH
   IF ERROR_NUMBER()=2627
      SET @msg='empno should not be duplicate'
   ELSE IF ERROR_NUMBER()=515
```

```
      SET @msg='name should not be null'
    ELSE IF ERROR_NUMBER()=547
      SET @msg='sal >= 3000'
    RAISERROR(@msg,16,1)
  END CATCH
```

01-oct-21

=> display list of errors

```
  SELECT * FROM sys.messages
```

how to add user define error to sys.messages table ?

```
 sp_addmessage  error code,severity level,error message
```

error code => should start from 50001

```
 sp_addmessage 50001,15,'sunday not allowed'

 DECLARE @eno int,@amt money
 SET @eno=7788
 SET @amt=1000
 IF DATENAME(dw,GETDATE())='SUNDAY'
    RAISERROR(50001,15,1)
 ELSE
   UPDATE emp SET sal=sal+@amt WHERE empno=@eno
```

NAMED TSQL BLOCKS :-

STORED PROCEDURES
STORED FUNCTIONS
TRIGGERS

SUB-PROGRAMS :-

STORED PROCEDURES
STORED FUNCTIONS

Advantages :-

1 modular programming :-

=> with the help of procedures & function a big tsql program can be divided into small modules

2 reusability :-

=> procedures & function can be stored in db and application which are connected to db can reuse these programs.

3 security :-

=> procedures and functions are stored in db and they are secured , only authorized users can execute these programs

4 invoked from front-end applications :-

=> procedures & functions can be called from front-end applications like java,.net,php etc.

5 improves performance :-

=> procedures improves performance because they are precompiled i.e. compiled already and ready for exectuion . when we create a procedure program is compiled and stored in db and whenever we call procedure only execution is repeated but not compilation.

STORED PROCEDURES :-

=> a procedure is a named TSQL block that accepts some input perform some action on db and may or may not returns a value.

=> procedures are created to perform one or dml operations on db.

syntax :-

```
CREATE OR ALTER PROCEDURE <NAME>
parameters if any
AS
  STATEMENTS
```

parameters :-

=> we can declare parameters and we can pass values to parameters
=> parameters are 2 types

1 INPUT
2 OUTPUT

INPUT :-

=> always recieves value
=> default
=> read only

OUTPUT :-

=> always sends value
=> write only

Example 1 :-

=> create procedure to increment specific employee sal by specific amount ?

```
 CREATE OR ALTER PROCEDURE update_salary
 @eno int,
 @amt money
 AS
  UPDATE emp SET sal=sal+@amt WHERE empno=@eno
```

Execution :-

1 ssms
2 another program
3 front-end application

executing from ssms :-

method 1 :- (positional)

```
EXECUTE update_salary 7499,1000
```

method 2 :- (named)

```
EXECUTE update_salary @eno=7369,@amt=1000
```

02-oct-21

OUTPUT parameter :-

=> create a procedure to increment specific employee sal by specific amount and after increment
   send the updated sal to calling program ?

```
CREATE OR ALTER PROCEDURE update_salary
@eno int,
@amt money,
@newsal money OUTPUT
AS
  UPDATE emp SET sal=sal+@amt WHERE empno=@eno
  SELECT @newsal=sal FROM emp WHERE empno=@eno
```

Execution :-

```
 DECLARE @s money
 EXECUTE update_salary 7499,1000,@s OUTPUT
 PRINT @s
```

```
 DECLARE @s money
 EXECUTE update_salary @eno=7499,@amt=1000,@newsal=@s OUTPUT
 PRINT @s
```

Example 3 :-

```
ACCOUNTS
ACCNO  ACTYPE  BAL
100    S       10000
100    S       20000

TRANSACTIONS
TRID   TTYPE   TDATE   TAMT    ACCNO

CREATE SEQUENCE S10
START WITH 1
INCREMENT BY 1
MAXVALUE 9999
```

create a procedure for money withdrawl ?

```
 CREATE OR ALTER PROCEDURE debit
 @acno int,
 @amt money,
 @newbal money OUTPUT
 AS
   DECLARE @bal money
   SELECT @bal=bal FROM accounts WHERE accno=@acno
   IF @amt > @bal
     RAISERROR('insufficient balance',15,1)
   ELSE
     BEGIN
        UPDATE accounts SET bal=bal-@amt  WHERE accno=@acno
        INSERT INTO transactions VALUES(NEXT VALUE FOR
S10,'W',GETDATE(),@amt,@acno)
        SELECT @newbal=bal FROM accounts WHERE accno=@acno
     END
```

create a procedure for money deposit ?

04-oct-21

create a procedure for money transfer ?

```
CREATE OR ALTER PROCEDURE transfer
@sacno int,
@tacno int,
@amt money
AS
  DECLARE @bal money,@cnt1 int,@cnt2 int
  SELECT @bal=bal FROM accounts WHERE accno=@sacno
  IF @amt > @bal
    RAISERROR('insufficient balance',15,1)
  ELSE
    BEGIN
       BEGIN TRANSACTION
       UPDATE accounts SET bal=bal-@amt WHERE accno=@sacno
       SET @cnt1=@@ROWCOUNT
       UPDATE accounts SET bal=bal+@amt WHERE accno=@tacno
       SET @cnt2=@@ROWCOUNT
       IF @cnt1=1 AND @cnt2=1
```

```
          COMMIT
        ELSE
           ROLLBACK
     END
```

Execution :-

```
EXECUTE transfer 100,101,1000
```

=> create procedure to insert record into emp44 table ?

```
CREATE TABLE emp44
(
 empno int primary key,
 ename varchar(10) not null,
 sal money check(sal>=3000)
)

CREATE OR ALTER PROCEDURE insert_emp44
@eno int,
@name varchar(10),
@sal money
AS
DECLARE @msg VARCHAR(100)
BEGIN TRY
INSERT INTO emp44 VALUES(@eno,@name,@sal)
END TRY
BEGIN CATCH
  IF ERROR_NUMBER()=2627
     SET @msg='empno should not be duplicate'
  ELSE IF ERROR_NUMBER()=515
    SET @msg='name should not be null'
  ELSE IF ERROR_NUMBER()=547
    SET @msg='sal >= 3000'
  RAISERROR(@msg,16,1)
END CATCH
```

USER DEFINE FUNCTIONS :-

=> functions created by user are called user define functions.

=> when predefin functions not meeting our requirements then we create our own functions called user define function

=> a function is also a named tsql block that accepts some input perform some calculation and must return a value.

=> function are created

1 for calculation
2 to fetch value from db

=> functions are 2 types

1 scalar valued functions
2 table valued functions

scalar valued functions :-

=> if function returns one value then it is called scalar valued function
=> return type of these function must be scalar types like int,varchar,date etc
=> return expression must be scalar variable.

```
 CREATE OR ALTER FUNCTION <NAME>(parameters if any) RETURN <type>
 AS
 BEGIN
   STATEMENTS
   RETURN <expr>
 END
```

Example 1 :-

```
 CREATE OR ALTER
    FUNCTION CALC(@a int,@b int,@op char(1)) RETURNS int
 AS
 BEGIN
    DECLARE @c int
    IF @op='+'
     SET @c=@a+@b
    ELSE IF @op='-'
     SET @c=@a-@b
    ELSE IF @op='*'
     SET @c=@a*@b
    ELSE
     SET @c=@a/@b
    RETURN @c
  END
```

Execution :-

1 sql commands
2 another pl/sql block
3 front-end applications

executing from sql commands :-

```
SELECT DBO.CALC(10,20,'*')   => 200

DECLARE @X INT
SET @X = DBO.CALC(10,20,'*')
PRINT @X
```

05-oct-21

=> create a function to calculate experience of particular employee ?

```
  CREATE OR ALTER FUNCTION expr(@eno int) RETURNS int
  AS
  BEGIN
     DECLARE @x int
     SELECT  @x=DATEDIFF(yy,hiredate,GETDATE()) FROM emp WHERE empno=@eno
     RETURN @x
 END
```

Execution :-

```
  SELECT DBO.EXPR(7369)   => 41

  SELECT EMPNO,ENAME,DBO.EXPR(EMPNO) AS EXPR FROM EMP
```

Assignment :-

1 create function to calculate tax ?

2 create function to calculate order amount of particular order ?

```
   input :- ordid = 1000
   output   amount = 7000
```

PRODUCTS

PRODID	NAME	PRICE
100		1000
101		2000
102		1500

ORDERS

ORDID	PRODID	QTY
1000	100	2
1000	101	1
1000	102	2
1001	100	2

TABLE VALUED FUNCTIONS :-

=> table valued functions returns records
=> return type must be TABLE
=> return expression must be select statement
=> table valued functions allows only one statement and it must be return statement
=> table valued functions are invoked in from clause.

syn :-

```
 CREATE OR ALTER FUNCTION <NAME>(parameters if any) RETURNS TABLE
 AS
  RETURN (SELECT STATEMENT)
```

Example 1 :-

create function to return list of employees working for specific dept ?

```
 CREATE OR ALTER FUNCTION getEmpList(@d int) RETURNS TABLE
 AS
  RETURN (SELECT * FROM emp WHERE deptno = @d)
```

Execution :-

```
 SELECT * FROM DBO.getEmpList(20)
```

Example 2 :-

create function to return top n employees list based on sal ?

```
 CREATE OR ALTER FUNCTION getTopNEmpList(@n int) RETURNS TABLE
 AS
  RETURN (SELECT *
          FROM (SELECT empno,ename,sal,
                       DENSE_RANK() over (ORDER BY sal DESC) as rnk
                 FROM emp) AS E
          WHERE rnk<=@n)


 SELECT * FROM DBO.getTopNEmpList(5)
```

system defined table valued function :-

```
  SELECT * FROM STRING_SPLIT('sachin ramesh tendulkar',' ')

 sachin
 ramesh
 tendulkar
```

=> difference between procedures & functions ?

	procedures	functions
1	may or may not returns a value	must return a value
2	can return multiple values	always returns one value
3	returns values using OUTPUT parameter	returns value using return statement
4	cannot be executed from sql commands	can be executed from sql commands
5	created to perform dml opertions	can't execute dml commands
6	created to perform some action on db	created for calculations
7	create procedure to update balance	create function to get balance

=> difference between scalar valued and table valued functions ?

scalar   table

1	returns one value	returns records
2	return type must be scalar types	return type must be table
3	return expression must a scalar variable	returns expression must be select stmt
4	invoked in select clause	invoked in FROM clause

Assignment :-

ACCOUNTS
ACCNO   NAME   ACTYPE BAL

TRANSACTIONS
TRID   TTYPE   TDATE   TAMT   ACCNO

```
CREATE SEQUENCE S10
START WITH 1
INCREMENT BY 1
MAXVALUE 9999
```

create procedures & functions to implement following bank transactions ?

1 account opening
2 account closing
3 money deposit
4 money withdrawl
5 money transfer
6 balance enquiry
7 particular customer statement between two given dates
8 latest N transaction of particular customer

06-oct-21

TRIGGERS :-

=> a trigger is also a named TSQL block like procedure but executed implicitly by sql server whenever user submits DML/DDL commands.

=> triggers are created

1 to control DMLs
2 to enforce complex rules and validations
3 to audit tables
3 to manage replicas
4 to generate values for primary key columns

syn :-

```
CREATE OR ALTER TRIGGER <NAME>
ON <TABNAME>
AFTER / INSTEAD OF  INSERT,UPDATE,DELETE
AS
BEGIN
    STATEMENTS
END
```

AFTER triggers :-

=> if trigger is AFTER then sql server will execute the trigger after executing DML

INSTEAD OF trigger :-

=> if trigger is INSTEAD OF then sql server executes the trigger instead of executing DML.

Example 1 :- create trigger to not to allo dmls on emp on sunday ?

```
CREATE OR ALTER TRIGGER T1
ON EMP
AFTER INSERT,UPDATE,DELETE
AS
BEGIN
    IF DATENAME(dw,GETDATE())='sunday'
      BEGIN
         ROLLBACK
         RAISERROR('sunday not allowed',15,1)
     END
END
```

Testing :- (getdate()=sunday)

```
  update emp set comm=500 where empno=7369 => ERROR
```

Example 2 :-

create trigger to not to allow dmls on emp table as follows ?

mon - fri	<10am and >4pm
sat	<10am and >2pm
sun	--------------

```
CREATE OR ALTER TRIGGER T2
ON EMP
AFTER INSERT,UPDATE,DELETE
AS
BEGIN
   IF DATEPART(dw,GETDATE()) BETWEEN 2 AND 6
      AND
      DATEPART(hh,GETDATE()) NOT BETWEEN 10 AND 15
       BEGIN
```

```
          ROLLBACK
          RAISERROR('only between 10am and 4pm',15,1)
        END
   ELSE IF DATEPART(dw,GETDATE())=7
           AND
           DATEPART(hh,GETDATE()) NOT BETWEEN 10 AND 13
        BEGIN
           ROLLBACK
           RAISERROR('only between 10am and 2pm',15,1)
        END
   ELSE IF DATEPART(dw,GETDATE())=1
      BEGIN
          ROLLBAK
          RAISERROR('sunday not allowed',15,1)
      END
END
```

Example 3 :-

create trigger to not to allow to update employee number and hiredate ?

```
 CREATE OR ALTER TRIGGER T3
 ON EMP
 AFTER UPDATE
 AS
 BEGIN
    IF update(empno) OR update(hiredate)
     BEGIN
         ROLLBACK
         RAISERROR('empno,hiredate cannot be updated',15,1)
    END
 END
```

08-oct-21

Magic Tables :-
----

 1 INSERTED
 2 DELETED

=> with the help of magic tables in triggers we can access data affected by dml
=> the record user is trying to insert is copied to inserted table
=> the record user is trying to delete is copied to deleted table
=> the record user is trying to update is copied to both inserted & deleted tables

```
  INSERT INTO emp(empno,ename,sal) VALUES(100,'A',4000)  =>  INSERTED
                                                             empno  ename
sal
                                                             100    A
4000

  DELETE FROM emp WHERE empno = 7499                    =>   DELETED
                                                             empno    ename
job     sal
```

```
                                                              7499     allen
salesman  1600

  UPDATE emp SET sal=2000 WHERE empno=7499              =>   DELETED
                                                           empno      sal
                                                           7499       1600

                                                           INSERTED
                                                           empno      sal
                                                           7499       2000
```

=> create trigger to not to allow to decrement salary ?

```
  CREATE OR ALTER TRIGGER T4
  ON EMP
  AFTER UPDATE
  AS
  BEGIN
     DECLARE @OLDSAL MONEY,@NEWSAL MONEY
     SELECT @OLDSAL=SAL FROM DELETED
     SELECT @NEWSAL=SAL FROM INSERTED
     IF @OLDSAL > @NEWSAL
       BEGIN
         ROLLBACK
         RAISERROR('sal cannot be decremented',15,1)
       END
 END
```

Testing :-

```
  UPDATE emp SET sal=1000 WHERE empno=7499  => ERROR
```

=> create trigger to insert details into emp_resign table when employee resigns from organization ?

```
 EMP_RESIGN
 EMPNO  ENAME   HIREDATE       DOR


 CREATE TABLE emp_resign
 (
   empno int,
   ename varchar(10),
   hiredate date,
   dor date
  )

 CREATE OR ALTER TRIGGER T5
 ON EMP
 AFTER DELETE
 AS
 BEGIN
   INSERT INTO emp_resign
   SELECT empno,ename,hriedate,getdate() FROM DELETED
 END
```

Testing :-

```
  DELETE FROM emp WHERE empno=7902


  SELECT * FROM EMP_RESIGN

  7902  FORD   1981-12-03     2021-10-08
```

INSTEAD OF trigger :-

=> if trigger is instead of then sql server executes the trigger instead of executing dml

```
EMP88
ENO     ENAME   SAL     COMM    HIREDATE

CREATE TABLE EMP88
(
  eno int,
 ename varchar(10),
 sal money,
 comm money,
 hiredate date
 )


CREATE OR ALTER TRIGGER T6
ON EMP88
INSTEAD OF INSERT
AS
BEGIN
  DECLARE @ENO INT,@NAME VARCHAR(10),@SAL MONEY,@COMM MONEY
  SELECT @NAME=ENAME,@SAL=SAL FROM INSERTED
  SELECT @ENO = ISNULL(MAX(ENO),0)+1 FROM EMP88
  SET @COMM=@SAL*0.1
  INSERT INTO EMP88 VALUES(@ENO,@NAME,@SAL,@COMM,GETDATE())
 END
```

TESTING :-

```
 INSERT INTO EMP88(ENAME,SAL) VALUES('A',5000)


 SELECT * FROM EMP88

 ENO    ENAME   SAL     COMM    HIREDATE
 1      A       5000    500     2021-10-08
```

10-oct-21

Auditing :-

=> Auditing means capturing changes made to table
=> Auditing means monitoring day-to-day actitivies on tables
=> changes are captured in some tables called audit tables
=> triggers are created to audit tables.

```
EMP_AUDIT
UNAME  OPERATION   OPTIME      NEW_ENO       NEW_ENAME      NEW_SAL
       OLD_ENO        OLD_ENAME      OLD_SAL
dbo    INSERT      ???         100           A              5000
       NULL           NULL           NULL
dbo    UPDATE      ???         100           A              6000
       100            A              5000
dbo    DELETE      ???         NULL          NULL           NULL
       100            A              6000
```

```
CREATE TABLE emp_audit
(
   uname      varchar(10),
   operation  varchar(10),
   optime     datetime,
   new_eno    int,
   new_ename  varchar(10),
   new_sal    money,
   old_eno    int,
   old_ename  varchar(10),
   old_sal    money
 )
```

create trigger to capture changes made to emp table ?

```
 CREATE OR ALTER TRIGGER T7
 ON EMP
 AFTER INSERT,UPDATE,DELETE
 AS
 BEGIN
     DECLARE @oldeno int,@oldename varchar(10),@oldsal money
     DECLARE @neweno int,@newename varchar(10),@newsal money
     DECLARE @cnt1 int,@cnt2 int,@op varchar(10)
     SELECT @oldeno=empno,@oldename=ename,@oldsal=sal FROM DELETED
     SELECT @neweno=empno,@newename=ename,@newsal=sal FROM INSERTED
     SELECT @cnt1=COUNT(*) FROM INSERTED
     SELECT @cnt2=COUNT(*) FROM DELETED
     IF @cnt1=1 AND @cnt2=0
        SET @op='INSERT'
     ELSE IF @cnt1=0 AND @cnt2=1
        SET @op='DELETE'
     ELSE
        SET @op='UPDATE'
     INSERT INTO emp_audit
VALUES(USER_NAME(),@op,GETDATE(),@neweno,@newename,@newsal,
                                           @oldeno,@oldename,@oldsal)
```

```
  END
```

Droping Trigger :-

```
 DROP TRIGGER T2
```

Dynamic SQL :-

=> SQL commands build at runtime are called dynamic SQL commands

```
   ex :- DROP TABLE emp (static sql command)

         DECLARE @TNAME VARCHAR(20)
         SET @TNAME='EMP'
         DROP TABLE @TNAME    (dynamic sql command)
```

=> Dynamic SQL is useful when we don't know tablenames and columnnames until runtime.

=> Dynamic SQL commands are executed by using

```
    1 EXEC procedure
    2 SP_EXECUTESQL procedure
```

using EXEC :-

=> dynamic sql command that you want to execute should be passed as a string to EXEC

```
               EXEC (' dynamic sql command ')
```

Example 1 :- create a procedure to drop table from db ?

```
   CREATE OR ALTER PROCEDURE drop_table
   @tname VARCHAR(20)
   AS
      DECLARE @str VARCHAR(100)
      SET @str = 'DROP TABLE ' + @TNAME
      EXEC(@str)
```

Example 2 :- create procedure to drop all tables from db ?

```
  CREATE OR ALTER PROCEDURE DROP_ALL_TABLES
  AS
   DECLARE C1 CURSOR FOR SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
   DECLARE @TNAME VARCHAR(30),@STR VARCHAR(30)
   OPEN C1
   FETCH NEXT FROM C1 INTO @TNAME
   WHILE(@@FETCH_STATUS=0)
   BEGIN
    SET @STR = 'DROP TABLE ' + @TNAME
    EXEC(@STR)
    FETCH NEXT FROM C1 INTO @TNAME
  END
    CLOSE C1
    DEALLOCATE C1
```

using SP_EXECUTESQL :-

Example 3 :-

=> write a prog to display no of rows in each and every table ?

```
  EMP  14
  DEPT 4
  CUST 10
```

```
DECLARE C1 CURSOR FOR SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
DECLARE @TNAME VARCHAR(30),  @Sqlcmd  NVARCHAR(1000) ,@cnt INT
OPEN C1
FETCH NEXT FROM C1 INTO @TNAME
WHILE(@@FETCH_STATUS=0)
BEGIN
 SET @Sqlcmd='SELECT @CNT=COUNT(*) FROM  ' + @TNAME
 EXEC sp_executesql @Sqlcmd , N'@cnt INT OUTPUT' ,@cnt=@cnt OUTPUT
 PRINT @TNAME +  '    '  + CAST(@cnt AS VARCHAR)
 FETCH NEXT FROM C1 INTO @TNAME
END
  CLOSE C1
  DEALLOCATE C1
```

11-oct-21

BACKUP & RESTORE :-

backup command :-

```
 USE MASTER
 BACKUP DATABASE [DB7PM] TO DISK = 'C:\DATA\DB7PM.BAK'
```

restore command :-

```
 USE MASTER
 RESTORE DATABASE [DB7PM] FROM DISK = 'C:\DATA\DB7PM.BAK'
```

procedure to take backup of all database :-

```
CREATE OR ALTER PROCEDURE backup_dbs
AS
DECLARE C1 CURSOR FOR select name from sys.databases
                                        where database_id > 4
DECLARE @name varchar(100),@fname varchar(100)
OPEN C1
FETCH NEXT FROM C1 INTO @name
```

```
WHILE(@@FETCH_STATUS=0)
BEGIN
   SET @fname = 'C:\DATA\' + @name + CONVERT(VARCHAR,GETDATE(),112) + '.bak"
   BACKUP DATABASE @name TO DISK = @fname
   FETCH NEXT FROM C1 INTO @name
END
   CLOSE C1
   DEALLOCATE C1
```