Database :-

=> a Database is a organized collection of interrelated data.
for example a univ db stores data related to students,courses
and faculty

Types of Databases :-
----

1 OLTP DB (online transaction processing)
2 OLAP DB (online analytical processing)

=> OLTP db is used for storing day-to-day transactions and OLAP db
used for analysis.

=> OLTP is for running business and OLAP is for to analyze business.

OLTP DB

CUST

CID	NAME	ADDR
1	A	MUM

OLAP DB

CUST

CID	NAME	ADDR	START	END
1	A	HYD	01	01
1	A	BLR	01	28/02
1	A	MUM	01/03	

=> day-to-day operations on db includes

C	create
R	read
U	update
D	delete

DBMS :- (Database Management System)

=> DBMS is a software used to create and to manage database.
=> DBMS is an interface between user and database.

USER-----------DBMS--------DB

Evolution of DBMS :-

1960 FMS (File Mgmt System)

1970 HDBMS (Hierarchical dbms)
NDBMS (Network dbms)

1980 RDBMS (Relational dbms)

1990 ORDMBS (Object Relational dbms)

RDBMS :-

=> RDBMS concepts introduced by E.F.CODD.
=> E.F.CODD introduced 12 rules called CODD rules.
=> a db software that supports all 12 rules called perfect rdbms.
=> according to E.F.CODD in rdbms data must be orgnized in tables
i.e. rows and columns.

```
 CUSTOMERS
 CID  NAME   ADDR => columns/fields/attributes
 1    A      HYD
 2    B      MUM
 3    C      DEL  => row/record/tuple
```

DATABASE = collection of tables
TABLE = collection of rows & cols
ROW = collection of field values
COLUMN = collection of values assigned to one field

=> every table must contain primary key to uniquely identify the
records.

ex :- accno,empid,aadharno,panno,voterid

RDBMS features :-
----

1 easy to access and manipulate data.
2 less redundency (duplication of data).
3 more security
4 gurantees data quality

5 supports data sharing
6 supports transactions

RDBMS softwares :-   (SQL databases)

SQL SERVER	from microsoft
ORACLE	from oracle corp
MYSQL	from oracle corp
DB2	from IBM
POSTGRESQL	from postgresql forum
RDS	from amazon

NoSQL databases :-
----

mongoDB
cassandra

29-OCT-22

ORDBMS :-

=> object relational dbms
=> It is a combination of RDBMS & OOPS

ORDBMS = RDBBMS + OOPS (reusability)

=> RDBMS doesn't support reusability but ORDBMS supports reusability

ORDBMS softwares :-

SQL SERVER
ORACLE
POSTGRESQL

what is db ?
what is dbms ?
what is rdbms ?
what is ordbms ?

DB Development Life Cycle :- (DBDLC)

Analysis
Design
Development
Testing
Implementation
Maintenance

=> DB is designed by db designers or architects by using

1. ER model (Entity Relationship)
2. Normalization

=> DB Development means creating tables inside the database
and DB is developed by developers and DBAs (Database Admin)

Developer	DBA
CREATING TABLES	INSTALLATION OF SQL SERVER
CREATING VIEWS	CREATING DATABASE
CREATING SYNONYMS	CREATING LOGINS
CREATING SEQUENCES	DB BACKUP & RESTORE
CREATING INDEXES	DB EXPORT & IMPORT
CREATING PROCEDURES	DB UPGRADATION & MIGRATION
CREATING FUNCTIONS	PERFORMANCE TUNING
CREATING TRIGGERS	
WRITING QUERIES	

SQL SERVER 2008 => 2019 UPGRADTION

MYSQL-------------------SQL SERVER MIGRATION

=> DB is tested by QA team (Quality Assurance) by using some tools
called testing tools for ex selenium.

=> Implementation means moving DB from DEV server to PROD server

==

31-oct-22 SQL SERVER

=> sql server is basically a rdbms product from microsoft
used to create and to manage database.

=> It can be used for both DB development & Administration

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

CLIENT / SERVER Architecture :-
----

1 SERVER
2 CLIENT

=> server is a system where sql server software is installed and
running and inside the server sql server manages database.

=> a client is also a system where users can

1 connects to server
2 sumbit requests to server
3 receives response from server

client tool :-

SSMS (SQL SERVER MGMT STUDIO)

How to connect to sql server :-

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

```
SERVER TYPE     :-  DB ENGINE
SERVER NAME     :-  DESKTOP-G2DM7GI
AUTHENTICATION  :-  SQL SERVER / WINDOWS
LOGIN           :-  SA  (SYSTEM ADMIN)
PASSWORD        :-  123
```

=> click CONNECT

USER-----SSMS---------------------SQL SERVER

CREATING DATABASE IN SQL SERVER :-

=> in Object Explorer select Databases => New Database

Enter Database Name :- DB4PM

=> click OK

=> a Database is created with following two files

1 DATA FILE (.MDF) (Master Data File)
2 LOG FILE (.LDF) (Log Data File)

Name	Type	Size	Autogrowth	Path
DB4PM	DATA	8	64	----
DB4PM_LOG	LOG	8	64	----

PATH :-

C:\Program Files\
Microsoft SQL Server\
MSSQL15.MSSQLSERVER\MSSQL\DATA\

DB4PM.MDF
DB4PM_LOG.LDF

USER-----SSMS--------------------SQL SERVER-------DB4PM

DOWNLOAD & INSTALL :-
-=------------------

sql server :-
----

download :-
----

https://www.microsoft.com/en-in/sql-server/sql-server-downloads

step-by-step installation :-
----

https://computingforgeeks.com/
install-sql-server-developer-edition-on-windows-server/

ssms :-

download

https://learn.microsoft.com/en-us/sql/ssms/
download-sql-server-management-studio-ssms?view=sql-server-ver16

01-nov-22

SQL :-

=> structured query language.
=> language used to communicate with sql server.
=> user communicates with sql server by sending commands called queries.
=> a query is command/instruction/question submitted to sql server
to perform some operation over db.
=> sql is common to all rdbms

SQL SERVER	ORACLE	MYSQL	POSTGRESQL
SQL	SQL	SQL	SQL

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

DDL (DATA DEFINITION LANG)
DML (DATA MANIPULATION LANG)
DQL (DATA QUERY LANG)
TCL (TRANSACTION CONTROL LANG)
DCL (DATA CONTROL LANG)

SQL

DDL	DML	DQL	TCL	DCL
creater	insert	select	commit	grant
alter	update		rollback	revoke
drop	delete		save transaction	
truncate	merge			

DATA & DATA DEFINITION :-

```
 EMPID ENAME      SAL      DATA DEFINITION/METADATA
   1    A          6000     DATA
```

```
USER----SSMS-----------SQL-------------SQL SERVER------DB
       tool          lang           software       storage
```

```
USER---SQLPLUS---------SQL--------------ORACLE--------DB
```

```
USER--MYSQLWORKBENCH------SQL---------------MYSQL------DB
```

Datatypes in sql server :-

=> Datatype specifies

1 type of the data allowed in a column
2 amount of memory allocated for column

DATATYPES

CHAR	INTEGER	FLOAT	CURRENCY	DATE	BINARY
CHAR	TINYINT	NUMERIC(P,S)	SMALLMONEY	DATE	BINARY
VARCHAR	SMALLINT		MONEY	TIME	VARBINARY
VARCHAR(MAX)	INT			DATETIME	VARBINARY(MAX)
	BIGINT				
NCHAR	NUMERIC(P)				
NVARCHAR					
NVARCHAR(MAX)					

CHAR(size) :-

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

ex :- NAME   CHAR(10)

```
VIJAY-----
      wasted
```

=> in char extra bytes are wasted so don't use char for variable length fields and use char for fixed length fields

```
GENDER  CHAR(1)

M
F

STATE_CODE  CHAR(2)

AP
TS
```

VARCHAR(size) :-

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

```
ex :-  NAME  VARCHAR(10)

      SACHIN----
             released
```

VARCHAR(MAX) :-

=> allows character data upto 2GB.

note :- CHAR/VARCHAR allows ascii chars (256 chars) that includes a-z,A-Z,0-9,special chars

```
PANNO      CHAR(10)
VEHNO      CHAR(10)
EMAILID    VARCHAR(20)
```

NCHAR/NVARCHAR/NVARCHAR(MAX) :-

N => National

=> allows unicode chars (65536 chars) this includes chars belongs to different languages.

Integer Types :-

=> allows numbers without decimal part

TINYINT	1 BYTE	0 TO 255
SMALLINT	2 BYTES	-32768 TO 32767
INT	4 BYTES	-2^31 TO 2^31-1 (-2147483648 to 2147483647)
BIGINT	8 BYTES	-2^63 TO 2^63-1

-9,223,372,036,854,775,808
to
9,223,372,036,854,775,807

```
AGE    TINYINT
EMPID  SMALLINT
```

02-nov-22

NUMERIC(P) :-

=> allows numbers without decimal part
=> allows numbers upto 38 digits

p => precision => no of digits => can be upto 38

```
EMPID   NUMERIC(4)

100
1000
10000  => NOT ALLOWED

AADHARNO  NUMERIC(12)

ACCNO     NUMERIC(13)
```

NUMERIC(P,S)/DECIMAL(P,S) :-

=> allows numbers with decimal part

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

```
   SALARY    NUMERIC(7,2)

   5000
   5000.50
  50000.50
 500000.50  => NOT ALLOWED

   5000.507  => ALLOWED  => 5000.51
   5000.503  => ALLOWED  => 5000.50
```

CURRENCY TYPES :-

=> used for fields related to money

```
 SMALLMONEY   4 BYTES  -214748.3648 to 214748.3647
 MONEY        8 BYTES  -922337203685477.5808
                        to
                       922337203685477.5807

       SALARY   SMALLMONEY
       BAL      MONEY
```

DATE & TIME :-

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

=> default date format in sql server is yyyy-mm-dd
=> default time format is HH:MI:SS

```
    EX :-  DOB    DATE
```

2003-10-15

LOGIN TIME

10:00:00

TXN_DT   DATETIME

2022-11-02 9:00:00

BINARY types :-

=> binary types allows multimedia object like audio,video,images

```
BINARY         => allows binary data upto 8000 bytes
VARBINARY      => allows binary data upto 8000 bytes
VARBINARY(MAX) => allows binary data upto 2GB

       PHOTO    BINARY(1000)  => extra bytes are wasted
       PHOTO    VARBINARY(1000) => extra bytes are released
```

CREATING TABLES IN SQL SERVER :-

```
CREATE TABLE <tabname>
(
   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 128 chars
4 table can have upto 1024 columns
5 no of rows unlimited

EMP123	VALID
123EMP	INVALID
EMP 123	INVALID
EMP*123	INVALID
EMP_123	VALID

Example :-

=> create table with following structure

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

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

sp_help :-   (SP => stored procedure)

=> command to see the structure of the table

```
SP_HELP  <tabname>
```

```
ex :- SP_HELP  EMP
```

```
    COLNAME   TYPE                  SIZE
       EMPID   tinyint            1
       ENAME            varchar                     10
       JOB       varchar                    10
       SAL       smallmoney             4
       HIREDATE  date                   3
```

INSERTING DATA INTO TABLE :-

=> "INSERT" command is used to insert data into table.
=> INSERT command creates a row
=> we can insert

1 single row
2 multiple rows

inserting single row :-

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

ex :-

```
 INSERT INTO emp VALUES(100,'sachin','clerk',5000,'2022-11-02')
```

inserting multiple rows :-

```
 INSERT INTO emp VALUES(101,'vijay','analyst',8000,GETDATE()),
                        (102,'rahul','manager',9000,'2020-04-15')
```

03-nov-22

inserting nulls :-

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

method 1 :-

```
 INSERT INTO emp VALUES(103,'kumar',NULL,NULL,'2019-05-12')
```

method 2 :-

```
 INSERT INTO emp(empid,ename,hiredate)
                VALUES(104,'ravi',GETDATE())
```

remaining two fields job,sal filled with nulls

Operators in SQL SERVER :-
----

1 Arithmetic Operators => + - * / %

10+5 => 15
10-5 => 5
10*5 => 50
10/5 => 2
10%5 => 0

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

10>5 => TRUE
10<5 => FALSE
10=5 => FALSE
10<>5 => TRUE

3 Logical Operators => AND OR NOT

rexpr1	AND	rexpr2	output
T	T	T	
T	F	F	
F	T	F	
F	F	F	

rexpr1	OR	rexpr2	output
T	T	T	
T	F	T	
F	T	T	
F	F	F	

NOT	rexpr	output
T	F	
F	T	

4 Special Operators => BETWEEN
IN
LIKE
IS
ANY
ALL

```
                    EXISTS
                    PIVOT

5  Set Operators      => UNION
                    UNION ALL
                    INTERSECT
                    EXCEPT
```

Displaying Data :-

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

```
  syn :- SELECT COLUMNS/*  FROM  TABNAME

        FROM clause     => specify tablename
        SELECT clause   => specify column names
               *      => 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
```

WHERE clause :-

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

```
  syn :- SELECT columns  FROM tabname [WHERE cond]
```

condition :-

=> condition is a relation expression

```
COLNAME  OP   VALUE
```

=> OP must be any relational operator like > >= < <=   = <>
=> if cond=true record is selected
=> if cond=false record is not selected

examples :-

=> display employee details whose id = 103 ?

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

=> employee details whose name = vijay ?

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

=> employee details earning more than 5000 ?

```
SELECT * FROM emp WHERE sal>5000
```

=> employees joined after 2020 ?

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

=> joined before 2020 ?

```
SELECT * FROM emp WHERE hiredate < '2020-01-01'
```

compound condition :-

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

```
 WHERE cond1  AND  cond2    OUTPUT
       T          T      T
       T          F      F
       F          T      F
       F          F      F
```

```
 WHERE cond1  OR  cond2    OUTPUT
```

T	T	T
T	F	T
F	T	T
F	F	F

=> display employees working as clerk,manager ?

```
   SELECT * FROM emp WHERE job='CLERK' OR job='MANAGER'
```

04-nov-22

=> employees working as clerk and earning more than 3000 ?

```
   SELECT * FROM emp WHERE job='clerk' AND  sal>3000
```

=> employees working for dept 30 and working as salesman
and earning more than 1500 ?

```
   SELECT *
   FROM EMP
   WHERE DEPTNO=30 AND JOB='SALESMAN' AND SAL>1500
```

=> display employees joined in 1981 year ?

```
   SELECT *
   FROM EMP
   WHERE HIREDATE >= '1981-01-01' AND  HIREDATE<='1981-12-31'
```

=> employees whose name = smith,blake ?

```
   SELECT * FROM emp WHERE ename='smith' OR  ename='blake'
```

scenario :-

STUDENTS

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

=> list of students who are passed ?

```
  SELECT * FROM STUDENTS WHERE S1>=35  AND S2>=35 AND S3>=35
```

=> list of students who are failed ?

```
  SELECT * FROM STUDENTS WHERE S1<35 OR  S2<35 OR S3<35
```

IN operator :-

=> use IN operator for list comparision

```
    ex :-  1,2,3,4,5
          'A','B','C','D'

      WHERE COLNAME IN (V1,V2,V3,----)
```

=> employees name = smith,blake,king ?

```
   SELECT * FROM EMP WHERE ENAME IN ('SMITH','BLAKE','KING')
```

=> employees working as clerk,manager,analyst ?

```
   SELECT * FROM EMP WHERE JOB IN ('CLERK','MANAGER','ANALYST')
```

=> not working for dept 10,20 ?

```
   SELECT * FROM EMP WHERE DEPTNO NOT IN (10,20)
```

SINGLE	MULTI
=	IN

BETWEEN operator :-

=> use BETWEEN operator for range comparision

```
   ex :- age between 20 and 40
         sal between 5000 and 1000

      WHERE COLNAME BETWEEN V1 AND V2
```

=> employees earning between 2000 and 5000 ?

```
SELECT * FROM EMP WHERE SAL BETWEEN 2000 AND 5000
```

=> employees joined in 1981 year ?

```
SELECT * FROM EMP WHERE HIREDATE BETWEEN '1981-01-01' AND '1981-12-31'
```

=> not joined in 1981 ?

```
SELECT *
FROM EMP
WHERE HIREDATE NOT BETWEEN '1981-01-01' AND '1981-12-31'
```

=> working as clerk,manager and earning between 2000 and 5000
and joined in 1981 and not working for dept 10,20 ?

```
SELECT *
FROM EMP
WHERE JOB IN ('CLERK','MANAGER')
     AND
     SAL BETWEEN 2000 AND 5000
     AND
     HIREDATE BETWEEN '1981-01-01' AND '1981-12-31'
     AND
     DEPTNO NOT IN (10,20)
```

scenario :-

PRODUCTS
prodid   pname   price   category   brand

=> display samsung,redmi,realme mobile phones price between 10000 and 20000 ?

```
SELECT *
FROM PRODUCTS
WHERE BRAND IN ('SAMSUNG','REDMI','REALME')
     AND
     CATEGORY='MOBILE PHONES'
     AND
     PRICE BETWEEN 10000 AND 20000
```

=>

CUST

CID NAME AGE GENDER CITY

1 display list of male customers ?
2 display male customers age between 20 and 40 ?
3 display male customers age between 20 and 40 and
staying in hyd,mum,del ?

```
SELECT *
FROM CUST
WHERE GENDER='M'
      AND
      AGE BETWEEN 20 AND 40
      AND
      CITY IN ('HYD','MUM','DEL')
```

LIKE operator :-

=> use LIKE operator for pattern comparision

ex :- name starts with 's'
name ends with 's'
name contains 'a'

```
WHERE COLNAME LIKE 'PATTERN'
```

=> pattern contains alphabets,digits,special chars and wildcard chars

wildcard chars :-

```
%   =>   0 OR MANY CHARS

_   =>   EXACTLY 1 CHAR
```

=> employees name starts with 'S' ?

```
SELECT * FROM EMP WHERE ENAME LIKE 'S%'
```

=> name ends with 'S' ?

```
SELECT * FROM EMP WHERE ENAME LIKE '%S'
```

=> name contains 'S' ?

```
SELECT * FROM EMP WHERE ENAME LIKE '%S%'
```

=> where 'a' is the 3rd char in their name ?

```
SELECT * FROM EMP WHERE ENAME LIKE '__A%'
```

05-nov-22

=> where 'E' is the 2nd char from last ?

```
SELECT * FROM EMP WHERE ENAME LIKE '%E_'
```

=> employees joined in jan month ? YYYY-MM-DD

```
SELECT * FROM EMP WHERE HIREDATE LIKE '_____01___'
```

=> employees joined in 1981 year ?

```
SELECT * FROM EMP WHERE HIREDATE LIKE '1981%'
```

Question :-

```
SELECT * FROM EMP WHERE JOB IN ('CLERK','%MAN%')
```

A ERROR
B RETURNS ONLY CLERK
C RETURNS CLERK,MANAGER
D NONE

ANS :- B

```
SELECT * FROM EMP WHERE SAL BETWEEN 5000 AND 2000
                              (WHERE SAL>5000 AND SAL<2000)
```

A ERROR
B RETURNS ROWS
C RETURNS NO ROWS
D NONE

ANS :- C

IS operator :-

=> use IS opertaor for NULL comparision

```
WHERE COLNAME IS NULL
```

=> employees not earning commission ?

```
SELECT * FROM EMP WHERE COMM IS NULL
```

=> employees earning commission ?

```
SELECT * FROM EMP WHERE COMM IS NOT NULL
```

summary :-

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

=> display ENAME ANNUAL SALARY ?

```
SELECT ENAME,SAL*12  AS ANNSAL
FROM EMP
```

=> display ENAME SAL HRA DA TAX   TOTSAL   ?

HRA = house rent allowance = 20% on sal
DA = dearness allowance   = 30% on sal
TAX = 10% on sal
TOTSAL = SAL + HRA + DA - TAX

```
SELECT ENAME,SAL,
       SAL*0.2 AS HRA,
       SAL*0.3 AS DA,
       SAL*0.1 AS TAX,
       SAL+(SAL*0.2)+(SAL*0.3)-(SAL*0.1) AS TOTSAL
FROM EMP

SMITH  800   160  240   80    1120
```

ORDER BY clause :-

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

```
SELECT columns
FROM tabname
[WHERE cond]
ORDER BY colname ASC/DESC,--------
```

=> default order is asc

=> arrange employee list name wise ascending ?

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

=> arrange list sal wise desc order ?

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

```
1  A  3000         2  B  5000
2  B  5000  ====>   4  D  4000
3  C  1000          1  A  3000
4  D  4000          3  C  1000
```

NOTE :-

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

```
SELECT empno,ename,sal,deptno
FROM emp
ORDER BY 3 DESC
```

=> above query sorts based on 3rd column in select list i.e. sal

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

```
SELECT empno,ename,sal,deptno
```

```
FROM  emp
ORDER BY deptno ASC,sal DESC

ORDER BY 4 ASC,3 DESC

1  A  3000 20            2  B  5000 10
2  B  5000 10            5  E  4000 10
3  C  4000 30  =====>     4  D  6000 20
4  D  6000 20            1  A  3000 20
5  E  4000 10            3  C  4000 30
```

07-NOV-22

=>

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

arrange students list total marks desc,m desc,p desc ?

```
SELECT *
FROM STUDENTS
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 70  50
```

=> but to display total in the output ?

```
SELECT * , M+P+C AS TOTAL,(M+P+C)/3 AS AVG
FROM STUDENTS
ORDER BY  M+P+C  DESC,M DESC,P DESC
```

DISTINCT :-

=> DISTINCT clause eliminates duplicates

```
DISTINCT col1,col2,-----
```

Ex :- SELECT DISTINCT job FROM emp

```
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

```
SELECT DISTINCT deptno FROM emp
```

```
10
20
30
```

TOP clause :-

=> used to display Top N rows from table

```
SELECT TOP <n>  columns/*
FROM tabname
[WHERE cond]
[ORDER BY col ASC/DESC]
```

=> display first 5 rows from emp table ?

```
SELECT TOP 5  * FROM emp
```

=> display top 5 employees based on salary ?

```
SELECT TOP 5  *
   FROM emp
   ORDER BY SAL DESC
```

=> display top 5 max salaries ?

```
SELECT DISTINCT TOP 5 sal
   FROM emp
ORDER BY sal DESC
```

=> display top 5 employees based on experience ?

```
SELECT TOP 5 *
FROM emp
ORDER BY hiredate ASC
```

==

DML commands :- (Data Manipulation Lang)

INSERT
UPDATE
DELETE
MERGE

=> all DML commands acts on table data.
=> by default all DML operations are auto committed (saved).
=> to stop this auto commit execute following command

```
SET IMPLICIT_TRANSACTIONS ON
```

=> to save the operation execute COMMIT command.
=> to cancel the operation execute ROLLBACK command.

UPDATE command :-

=> update command used to modify the table data.
=> we can update all rows and specific rows.
=> we can update single column or multiple columns.

```
UPDATE <tabname>
SET colname = value , colname = value,------------
[WHERE cond]
```

examples :-

=> update all the employees comm with 500 ?

```
UPDATE EMP SET COMM = 500
```

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

```
UPDATE EMP SET COMM = 500 WHERE COMM IS NULL
```

=> update sal to 2000 and comm to 800 whose empno = 7369 ?

```
UPDATE EMP
SET SAL = 2000 , COMM = 800
WHERE EMPNO = 7369
```

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

```
UPDATE EMP
SET SAL =  SAL + (SAL*0.2) , COMM = COMM + (COMM*0.1)
WHERE JOB='SALESMAN'
      AND
      HIREDATE LIKE '1981%'
```

=> transfer all the employees from 10th to 40th dept ?

```
UPDATE EMP SET DEPTNO = 40  WHERE DEPTNO = 10
```

08-nov-22

DELETE :-

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

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

ex :-   delete all rows from emp ?

```
DELETE FROM EMP
```

delete employees joined in 1980 ?

```
DELETE FROM EMP WHERE HIREDATE LIKE '1980%'
```

DDL commands :-

CREATE
ALTER
DROP
TRUNCATE

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

=> all DDL commands are auto committed (saved).

=> execute the following command to stop auto commit

```
 SET IMPLICIT_TRANSACTIONS ON
```

ALTER command :-

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

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

Adding a column :-

```
 ALTER TABLE <tabname>
      ADD colname datatype(size),----------
```

ex :- add column gender to emp table ?

```
 ALTER TABLE EMP
      ADD GENDER CHAR(1)
```

=> after adding by default the new column is filled with null values to insert data into the new column use UPDATE command.

```
UPDATE EMP SET GENDER='M'  WHERE EMPNO=7499
```

Droping a column :-

```
ALTER TABLE <tabname>
    DROP COLUMN  col1,col2,-----
```

ex :-

=> drop columns gender,comm ?

```
 ALTER TABLE emp
     DROP COLUMN gender,comm
```

Modifying column :-

```
 ALTER TABLE <tabname>
     ALTER COLUMN colname datatype(size)
```

=> increase the size of ename to 20 ?

```
  ALTER TABLE EMP
     ALTER  COLUMN  ENAME VARCHAR(20)

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

=> change the datatype of empno to smallint ?

```
  ALTER TABLE EMP
      ALTER COLUMN EMPNO SMALLINT
```

DROP command :-

=> used to drop table from database.
=> 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

syn :- TRUNCATE TABLE <tabname>

EX :- TRUNCATE TABLE EMP

=> sql server goes to memory and releases all the pages allocated for table and when pages are released then data stored in the pages also deleted.

DELETE VS TRUNCATE :-

	DELETE	TRUNCATE
1	DML	DDL
2	can delete all rows or specific rows	can delete only all rows but cannot 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 tabname','new tabname'

ex :- rename table emp to employees ?

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

rename column hiredate to doj ?

```
SP_RENAME 'EMPLOYEES.HIREDATE','DOJ'
```

09-nov-22

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 :-

GETDATE() :-

=> returns current date & time

```
SELECT GETDATE()  => 2022-11-09 16:25:42.427
```

DATEPART() :-

=> used to extract part of the date

DATEPART(interval,date)

```
SELECT DATEPART(yy,GETDATE())  => 2022
           mm           => 11
           dd           => 9
           dw           => 4  (day of the week)

                         1  sunday
                         2  monday

                         7  saturday
           dayofyear     =>  313 (out of 365 days)
           qq           =>  4   (quarter)

                         1  jan-mar
                         2  apr-jun
                         3  jul-sep
                         4  oct-dec
           hh           =>  hour part
           mi           =>  minutes
           ss           =>  seconds
```

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

```
  SELECT *
  FROM EMP
  WHERE DATEPART(MM,HIREDATE) IN (1,4,12)
```

=> employees joined in leap year ?

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

=> employees joined on sunday ?

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

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

```
  SELECT *
```

```
 FROM EMP
 WHERE DATEPART(YY,HIREDATE) = 1981
       AND
       DATEPART(QQ,HIREDATE) = 2
```

DATENAME() :-

=> used to extract part of the date

```
  DATENAME(interval,date)
```

	MM	DW
DATEPART	11	4
DATENAME	NOVEMBER	WEDNESDAY

display ENAME JOIN_DAY ?

```
SELECT ENAME,DATENAME(DW,HIREDATE) AS DAY
FROM EMP


SELECT ENAME,DATEPART(YY,HIREDATE) AS YEAR,
          DATENAME(MM,HIREDATE) AS MONTH,
                          DATEPART(DD,HIREDATE) AS DAY
FROM EMP
```

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

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

DATEDIFF() :-

=> used to find different between two dates

```
  DATEDIFF(interval,start date,end date)
```

```
 SELECT DATEDIFF(yy,'2021-11-09',GETDATE())  =>   1
                mm                        =>   12
                dd                      =>   365
```

=> display ENAME EXPERIENCE in years ?

```
  SELECT ENAME,
         DATEDIFF(YY,HIREDATE,GETDATE()) AS EXPERIENCE
  FROM EMP
```

=> display ENAME EXPERIENCE ?
                  M YEARS N MONTHS

```
  EXPERIENCE = 40 MONTHS = 3 YEARS 4 MONTHS

  YEARS =  MONTHS/12  = 40/12 = 3

  MONTHS = MONTHS%12  = 40%12 = 4
```

```
  SELECT ENAME,
         DATEDIFF(MM,HIREDATE,GETDATE())/12 AS YEARS,
         DATEDIFF(MM,HIREDATE,GETDATE())%12 AS MONTHS
  FROM EMP
```

DATEADD() :-

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

```
  DATEADD(interval,int,date)

  SELECT DATEADD(yy,1,getdate())   =>  2023-11-09
  SELECT DATEADD(dd,10,getdate())  =>  2022-11-19
  SELECT DATEADD(mm,-2,getdate())  =>  2022-09-09
```

FORMAT() :-

=> used to display dates in different formats

```
      FORMAT(date,'format')

  SELECT FORMAT(GETDATE(),'dd.MM.yyyy') => 10.11.2022
```

dd => day
MM => month
yyyy => year
hh => hour part
mm => minutes
ss => seconds

=> display ENAME HIREDATE ?

display hiredate in mm/dd/yyyy format ?

```
SELECT ENAME,
      FORMAT(HIREDATE,'MM/dd/yyyy') as hiredate
FROM EMP
```

scenario :-

```
INSERT INTO EMP(EMPNO,ENAME,SAL,HIREDATE)
      VALUES(888,'ABC',4000,GETDATE())
```

=> display list of employees joined in today ?

```
SELECT *
FROM EMP
WHERE HIREDATE = GETDATE()  => NO ROWS

      2022-11-10 =  2022-11-10 16:33:20
```

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

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

Question :-

GOLD_RATES
DATEID   RATE

```
2018-01-01   ???
2018-01-02   ???

2022-11-10   ???
```

1 display today's gold rate ?
2 display yesterday's gold rate ?
3 display last month same day gold rate ?
4 display last year same day gold rate ?

```
1  SELECT * FROM GOLD_RATES
        WHERE DATEID = FORMAT(GETDATE(),'yyyy-MM-dd')

2  SELECT * FROM GOLD_RATES
       WHERE DATEID = FORMAT(DATEADD(dd,-1,GETDATE()),'yyyy-MM-dd')

3   SELECT * FROM GOLD_RATES
       WHERE DATEID = FORMAT(DATEADD(mm,-1,GETDATE()),'yyyy-MM-dd')

4  SELECT * FROM GOLD_RATES
       WHERE DATEID = FORMAT(DATEADD(yy,-1,GETDATE()),'yyyy-MM-dd')
```

EOMONTH() :-

=> returns end of month i.e. last day of the month

```
     EOMONTH(date,int)

  SELECT EOMONTH(GETDATE(),0)   => 2022-11-30
                         1    => 2022-12-31
                        -1    => 2022-10-31
```

1 display first day current month ?
2 display first day of the next month ?
3 display first day of the current year ?
4 display first day of the next year ?

STRING functions :-

UPPER() :-

=> converts string to uppercase

```
    UPPER(arg)

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

LOWER() :-

=> converts string to lowercase

```
    LOWER(arg)

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

display ENAME SAL ? display names in lowercase ?

```
SELECT LOWER(ENAME) AS ENAME,SAL FROM EMP
```

=> convert names to lowercase in table ?

```
  UPDATE EMP SET ENAME = LOWER(ENAME)
```

LEN() :-

=> returns string length i.e. no of chars.

```
   LEN(arg)

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

=> employees name contains 5 chars ?

```
   SELECT * FROM EMP WHERE ENAME LIKE '_____'

   SELECT * FROM EMP WHERE LEN(ENAME)=5
```

=> arrange employee names based on length ?

```
  SELECT ename,len(ename) as length
  FROM EMP
  ORDER BY len(ename) asc
```

11-nov-22

LEFT() :-

=> used to extract chars from left side

```
    LEFT(string,len)

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

=> employees name sarts with 's' ?

```
   SELECT * FROM emp WHERE ename LIKE '

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

RIGHT() :-

=> used to extract chars from right side

```
        RIGHT(string,len)

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

=> employees name ends with vowel ?

```
   SELECT * FROM EMP
      WHERE RIGHT(ename,1) IN ('a','e','i','o','u')

   SELECT * FROM EMP
      WHERE ENAME LIKE '%[aeiou]'
```

=> employees name starts and ends with same char ?

```
   SELECT *
   FROM EMP
   WHERE ENAME LIKE  'A%A'
         OR
         ENAME LIKE 'B%B'
   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
  SELECT SUBSTRING('hello welcome',10,3)   => com
```

scenario :-

generate emailids as follows ?

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

'a' + 'b' => ab

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

=> store emailid in db ?

STEP 1 :- add emailid column to emp table

```
 ALTER TABLE EMP
      ADD EMAILID VARCHAR(30)
```

STEP 3 :- update column with emailids

```
 UPDATE EMP
 SET EMAILID = LEFT(ename,3) + LEFT(empno,3) + '@tcs.com'
```

REPLICATE() :-

=> repeats character for given no of times

```
   REPLICATE(char,len)

  SELECT REPLICATE('*',5)   => *****
```

```
 => display  ENAME   SAL   ?
                     ******
                     *******
```

```
   SELECT ENAME,REPLICATE('*',LEN(sal)) AS SAL FROM EMP
```

scenario :-

ACCOUNTS
ACCNO
123456789456

your a/c no XXXX9456 debited ------

```
  REPLICATE('X',4) + RIGHT(ACCNO,4)
```

REPLACE() :-

=> used to replace one string with another string

```
    REPLACE(str1,str2,str3)

  SELECT REPLACE('hello','ell','abc')  => habco
  SELECT REPLACE('hello','l','abc')    => heabcabco
  SELECT REPLACE('hello','elo','abc')  => hello
  SELECT REPLACE('@@he@@ll@o@','@','') => hello
```

TRANSLATE() :-

=> used 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 i.e. converting plain text to cipher text.

```
   SELECT ENAME,
          TRANSLATE(SAL,'0123456789.','*pK%t$@#^&!') as sal
   FROM EMP

   jones  2975.00   => K&#$!**
```

Question :-

remove all the special chars from @#he*&ll^%o$?

```
    TRANSLATE('@#he*&ll^%o$','@#*&^%$','*******') => **he**ll**o*

   SELECT
   REPLACE(TRANSLATE('@#he*&ll^%o$','@#*&^%$','*******'),'*','')
```

O/P :- hello

12-nov-22

CHARINDEX() :-

=> returns position of a char in a string

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

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

SCENARIO :-

```
CUST
CID    NAME
10     SACHIN TENDULKAR
11     ROHIT SHARMA
```

output :-

```
  CID     FNAME         LNAME
  10      SACHIN        TENDULKAR
  11      ROHIT         SHARMA
```

```
 FNAME  =  SUBSTRING(STRING,START,LENGTH)

           SUBSTRING(NAME,1,CHARINDEX(' ',NAME)-1)

 LNAME  =  SUBSTRING(NAME,CHARINDEX(' ',NAME)+1,LEN(NAME))
```

```
  SELECT CID,
         SUBSTRING(NAME,1,CHARINDEX(' ',NAME)-1) AS FNAME,
         SUBSTRING(NAME,CHARINDEX(' ',NAME)+1,LEN(NAME)) AS LNAME
  FROM CUST
```

MATHEMATICAL FUNCTIONS :-

ABS() :- returns absolute value

```
  ABS(NUMBER)
```

```
 SELECT ABS(-10)   =>  10
        ABS(10)    =>  10
```

POWER() :- used to calculate power

```
   POWER(num1,num2)
```

```
 SELECT POWER(3,2) => 9
```

SQRT() :- returns square root

```
  SQRT(number)
  SELECT SQRT(16)  => 4
```

SIGN() :- returns whether expr is positve or negative

```
  SIGN(number)

 SELECT SIGN(10)  => 1
        SIGN(-10) => -1
        SIGN(0)   => 0
```

rounding numbers :-

```
  ROUND
  CEILING
  FLOOR
```

ROUND :-

=> used to round number to integer or to decimal places based on avg

```
   38.567845   => 39
                  38.57
                  38.5678

        ROUND(number,decimal places)

  SELECT ROUND(38.567845,0)    =>  39

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

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

   SELECT ROUND(38.567845,2)   => 38.57
   SELECT ROUND(38.567845,4)   => 38.5678

   SELECT ROUND(386,-2)        => 400

   300----------------350----------------400
   SELECT ROUND(386,-1)        => 390

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

   SELECT ROUND(386,-3)        =>  0

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

  SELECT ROUND(4567,-1),ROUND(4567,-2),ROUND(4567,-3)

        4570          4600          5000

 CEILING() :-
 ------------

 => rounds number always to highest

  CEILING(number)

  SELECT CIELING(3.1)   => 4

 FLOOR() :-
 ---------

  => rounds number always to lowest

    FLOOR(number)

  SELECT FLOOR(3.9)   =>   3

 CONVERSION FUNCTIONS :-
 ----------------------

 => used to convert one type to another type
 => sql server provided 2 functions for conversion

  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
```

=> display smith joined on 1980-12-17 as clerk ?

```
 SELECT ename + ' joined on ' + CAST(hiredate AS VARCHAR)
                  + ' as ' + job FROM emp
```

CONVERT() :-

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

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

diff b/w cast & convert ?

=> using convert we can display dates & money in different formats
   which is not possible using cast function.

Displying dates in different formats :-

```
  CONVERT(VARCHAR,DATE,STYLE-NUMBER)

  SELECT CONVERT(VARCHAR,GETDATE(),101)    =>  11/12/2022

                                    104     =>  12.11.2022

                                    114    =>   17:31:45:030
```

14-nov-22

Displaying Money in different formats :-

```
 CONVERT(VARCHAR,MONEY,STYLE-NUMBER)
```

STYLE-NUMBERS

0 => 2 digits after decimal

1 => displays thousand seperator

2 => 4 digits after decimal

=> display ENAME SAL ?
display salaries with thousand seperator ?

```
SELECT ENAME,CONVERT(VARCHAR,SAL,1) AS SAL FROM EMP
```

```
SELECT  CONVERT(VARCHAR,CAST(5000 AS MONEY),1) => 5,000.00
```

SPECIAL FUNCTIONS :-

ISNULL() :-

=> used to convert null values

```
ISNULL(arg1,arg2)
```

if arg1 = null returns arg2

if arg1 <> null returnsa 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   NULL    800
      ALLEN  1600  300     1900
```

display ename sal comm ?
display if comm = null display N/A ?

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

COALESCE() :-

=> returns first not null expression

```
   COALESCE(arg1,arg2,arg3,---)

 SELECT COALESCE(100,200,300)     => 100

  SELECT COALESCE(NULL,300,200)   => 300
```

scenario :-

```
 CUST
 CID   NAME   ADDR1   ADDR2
 1     A      NULL    HYD
 2     B      MUM     NULL
 3     C      BLR     HYD

 SELECT CID,NAME,COALESCE(ADDR1,ADDR2) AS ADDR FROM CUST

 1  A   HYD
 2  B   MUM
 3  C   BLR
```

Analytical Functions :-

RANK & DENSE_RANK :-

=> used to find ranks.

=> ranking is always based on some column.
=> for rank functions data must be sorted

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

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

```
SELECT ENAME,SAL,
     OVER (ORDER BY SAL DESC) AS RNK
FROM EMP

SELECT 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 in 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

PARTITION BY clause :-

=> used to find ranks with in group , for example to find ranks with dept first we need to divide the table dept wise and apply rank/dense_rank function on each dept instead of applying on whole table.

ex :- DENSE_RANK() OVER (

```
                          PARTITION BY deptno
                          ORDER BY sal DESC
                         )
```

```
SELECT ENAME,SAL,DEPTNO,
     DENSE_RANK() OVER (PARTITION BY DEPTNO
                        ORDER BY SAL DESC) AS RNK
FROM EMP
```

```
     KING   5000.00       10      1
     CLARK           2450.00       10      2
     MILLER          1300.00       10      3

     FORD  3000.00       20      1
     SCOTT           3000.00       20      1
     JONES           2975.00       20      2
     ADAMS           1100.00       20      3
```

16-nov-22

ROW_NUMBER() :-

=> returns record numbers
=> row_number is also based on some column
=> for row_number data must be sorted

```
    ROW_NUMBER() OVER (ORDER BY COL ASC/DESC)
```

example :-

```
SELECT ename,sal,
    ROW_NUMBER() OVER (ORDER BY sal DESC) as rno
FROM emp
```

SAL	RNK	DRNK	RNO
5000	1	1	1
4000	2	2	2
3000	3	3	3
3000	3	3	4
3000	3	3	5
2000	6	4	6

2000	6	4	7
1000	8	5	8

AGGREGATE FUNCTIONS :-

=> these functions are called aggregate functions because
they 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  => 1983-01-12
SELECT MAX(ename)  FROM emp    => WARD
```

MIN() :-

=> returns minimum value

MIN(arg)

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

SUM() :-

=> returns total

SUM(arg)

```
SELECT SUM(sal) FROM emp  => 29025.00
```

=> round the total sal to hundreds ?

```
SELECT ROUND(SUM(sal),-2) FROM EMP  => 29000
```

29000-----------29050----------------29100

=> after rounding it to hundreds display with thousand seperator ?

```
SELECT
  CONVERT(VARCHAR,CAST(ROUND(SUM(sal),-2) AS MONEY),1)
FROM EMP

O/P :- 29,000
```

AVG() :-

=> returns average value

```
    AVG(expr)
```

```
SELECT AVG(sal) FROM emp  => 2073.214285
```

=> round avg(sal) to lowest integer ?

```
SELECT FLOOR(AVG(sal)) FROM emp => 2073
```

NOTE :- sum,avg functions cannot be applied on char,date columns
can be applied only on numeric columns

```
SUM(hiredate)  => ERROR
AVG(hiredate)  => ERROR
```

COUNT(*) :-

=> returns no of rows in a table

```
SELECT COUNT(*) FROM EMP  =>  14
```

=> how many employees joined on sunday ?

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

=> how many employees joined in year 1981 ?

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

note :-

=> aggregate functions are not allowed in where clause and they are allowed only in select,having clauses.

```
SELECT ENAME FROM EMP WHERE SAL = MAX(SAL)  => error

SELECT ENAME FROM EMP WHERE COUNT(*) = 3    => error
```

summary :-

DATE :- getdate(),datepart,datename,datediff,dateadd,eomonth

CHAR :- upper,lower,len,left,right,substring,charindex,
replicate,replace,translate

NUMERIC :- abs,power,sqrt,sign,round,ceiling,floor

CONVERSION :- cast,convert

SPECIAL :- isnull,coalesce

ANALYTICAL :- rank,dense_rank,row_number

AGGREGATE :- max,min,sum,avg,count(*)

using aggregate functions as analytical functions :-
----

diff b/w aggregate & analytical functions ?

=> aggregate functions returns one value from the group of rows

=> analytical functions returns one value for each row.

Example :-

sum as aggregate :-

```
SELECT SUM(sal) FROM emp  => 29025
```

sum as analytical :-

```
 SELECT empno,ename,sal,
        SUM(sal) OVER (ORDER BY empno ASC) as running_total
 FROM emp
```

```
 7369  smith   800    800
 7499  allen   1600   2400
 7521  ward     1250   3650
```

NOTE :-

```
 SELECT COL1,COL2 FROM TABNAME
```

no of values return by col1 = no of values return by col2

```
 SELECT ename,MAX(sal) FROM emp  =>  ERROR
        ----- --------
        14     1

  SELECT MIN(sal),MAX(sal) FROM emp  => SUCCESSFUL
         ------   --------
          1        1

  SELECT ename,SUM(sal) OVER (order by empno asc) as total FROM emp =>
         ------ --------------------------------------------
           14     14
```

17-nov-22

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 , for ex to calculate job wise no of employees or dept wise total salary etc.

```
     EMP
     EMPNO  ENAME  SAL   DNO
```

```
1         A        5000 10
2         B        3000  20      GROUP BY             10    8000
3         C        4000  30  ================>      20     7000
4         D        3000  10                          30    4000
5         E        4000   20

 detailed data                                     summarized data
```

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

syntax :-

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

Execution :-

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

examples :-

=> display dept wise total salary ?

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

```
 10  8750
 20  10875
 30  9400
```

FROM EMP :-

EMP

EMPNO	ENAME	SAL	DNO
1	A	5000	10
2	B	3000	20
3	C	4000	30
4	D	3000	10
5	E	4000	20

GROUP BY deptno :-

```
10
      1   A   5000
      4   D  3000

20
      2    B   3000
      5    E   4000

30
      3   C    4000
```

SELECT deptno,SUM(sal) as totsal :-

```
10      8000
20      7000
30      4000
```

=> display no of employees for each job ?

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

```
ANALYST                   2
CLERK                     4
MANAGER      3
PRESIDENT    1
SALESMAN     4
```

=> display year wise no of employees joined ?

```
SELECT DATEPART(yy,hiredate) as year,COUNT(*) as cnt
FROM EMP
GROUP BY  DATEPART(yy,hiredate)

    1980    1
    1981    10
    1982    2
    1983    1
```

=> month wise no of employees joined in in the year 1981 ?

```
SELECT DATENAME(mm,hiredate) as month,COUNT(*) as cnt
FROM EMP
WHERE DATEPART(yy,hiredate)=1981
GROUP BY  DATENAME(mm,hiredate)
```

=> display departments having more than 3 employees ?

```
SELECT deptno,COUNT(*) as cnt
FROM EMP
WHERE COUNT(*) > 3
GROUP BY deptno              => ERROR
```

NOTE :- sql server cannot calculate dept wise count before group by and it can can calculate only after group by , so apply the condition COUNT(*) > 3 after group by using HAVING clause.

```
SELECT deptno,COUNT(*) as cnt
FROM EMP
GROUP BY deptno
HAVING COUNT(*) > 3

20     5
30      6
```

WHERE VS HAVING :-

	WHERE	HAVING
1	selects specific rows	selects specific groups

2	applied before group by	applied after group by
3	use where clause if condition doesn't contain aggregate function	use having clause if condition contains aggregate function

18-nov-22

scenario :-

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

=> find southern states having more than 5cr population ?

```
SELECT  state,COUNT(*) as cnt
FROM persons
WHERE  state  IN ('AP','TS','KL','KA','TN')
GROUP BY state
HAVING COUNT(*) > 50000000
```

Grouping based on multiple columns :-

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

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

    10    CLERK                1300
          MANAGER  2450
          PRESIDENT            5000

   20  ANALYST                 6000
       CLERK                   1900
       MANAGER     2975
```

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

```
 SELECT state,gender,COUNT(*)
 FROM persons
 GROUP BY state,gender
 ORDER BY state

  AP      MALE       ?
          FEMALE   ?

 AR      MALE      ?
          FEMALE ?

AS       MALE    ?
         FEMALE ?
```

Question :-

EMP

ENO	ENAME	SAL
1	A	5000
2	B	3000
3	C	4000
1	A	5000
2	B	3000

=> display duplicate records ?

```
SELECT eno,ename,sal
FROM emp
GROUP BY eno,ename,sal
HAVING COUNT(*) > 1

1       A            5000
2       B           3000
```

1 eliminate duplicates in select stmt output ?   => distinct
2 find duplicate rows ?   => group by   having count(*) > 1
3 remove duplicate rows from table ?   => subqueries

ROLLUP & CUBE :-

=> both functions are used to calculate subtotal and grand total.

```
    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 deptno ASC

   NULL     NULL    29025   =>  grand total

  10          CLERK        1300
              MANAGER   2450
             PRESIDENT  5000
                           8750   => subtotal
```

CUBE :-

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

```
   SELECT deptno,job,SUM(sal) as totsal
   FROM emp
   GROUP BY CUBE(deptno,job)
   ORDER BY deptno ASC ,job ASC

      NULL  NULL            29025.00  => grand total
      NULL  ANALYST                6000.00    => job subtotal
      NULL  CLERK                  4150.00    => job subtotal

      10      NULL            8750.00   => dept subtotal
      10      CLERK                  1300.00
      10      MANAGER     2450.00
      10      PRESIDENT   5000.00
```

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

```
SELECT state,gender,COUNT(*)
FROM persons
GROUP BY ROLLUP(state,gender)
ORDER BY state ASC
```

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

```
SELECT state,gender,COUNT(*)
FROM persons
GROUP BY CUBE(state,gender)
ORDER BY state ASC,gender ASC
```

===

CASE statement :-

=> used to implement if-then-else.
=> use case statement to return values based on condition.
=> case statements are 2 types

1 simple case
2 searched case

simple case :-

=> use simple case when condition based on "=" operator.

```
CASE EXPR/COLNAME
WHEN VALUE1 THEN RETURN EXPR1
WHEN VALUE2 THEN RETURN EXPR2
-------------------------------
ELSE RETURN EXPR
END
```

=> display   ENAME SAL   DNAME   ?

```
if deptno=10 display  HR
          20 display  IT
          30 display  SALES
          else         OTHER
```

```
SELECT ENAME,SAL,
          CASE  DEPTNO
          WHEN 10 THEN  'HR'
          WHEN 20 THEN 'IT'
          WHEN 30 THEN 'SALES'
          ELSE 'OTHER'
          END
FROM EMP
```

19-nov-22

searched case :-

=> use searched case when condition not based on "=" operator.

```
CASE
WHEN COND1 THEN RETURN EXPR1
WHEN COND2 THEN RETURN EXPR2
------------------------
ELSE RETURN EXPR
END
```

=> display ENAME   SAL   SALRANGE   ?

```
if sal>3000 display Hisal
  sal<3000 display Losal
  otherwise            Avgsal
```

```
SELECT ENAME,SAL,
          CASE
          WHEN SAL>3000 THEN 'Hisal'
          WHEN SAL<3000 THEN 'Losal'
          ELSE  'Avgsal'
          END AS SALRANGE
 FROM EMP
```

```
     SMITH   800    Losal
     SCOTT  3000  Avgsal
     KING     5000  Hisal
```

=> display SNO   SNAME   TOTAL   AVG RESULT ?

STUDENT

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

```
 SELECT SNO,SNAME,
               S1+S2+S3 AS TOTAL,
               (S1+S2+S3)/3 AS AVG,
                CASE
                WHEN S1>=35 AND S2>=35 AND S3>=35 THEN 'PASS'
                ELSE 'FAIL'
                END AS RESULT
FROM STUDENT
```

example for range grouping :-

```
 SELECT CASE
          WHEN SAL BETWEEN 1 AND 2000 THEN '1-2000'
          WHEN SAL BETWEEN 2001 AND 4000 THEN '2001-4000'
          WHEN SAL>4000 THEN 'ABOVE 4000'
          END AS SALRANGE ,COUNT(*) AS CNT
FROM EMP
GROUP BY  CASE
              WHEN SAL BETWEEN 1 AND 2000 THEN '1-2000'
              WHEN SAL BETWEEN 2001 AND 4000 THEN '2001-4000'
              WHEN SAL>4000 THEN 'ABOVE 4000'
              END
```

Question :-

PERSONS

NAME	GENDER	AGE	ADDR	CITY	STATE

=> display age group wise no of persons ?

```
1-20    ?
21-40   ?
41-60   ?
>60      >
```

==

Integrity Constraints :-

=> Integrity Constraints are rules to maintain data integrity i.e. data quality.

=> used to prevent users from entering invalid data.

=> used to enforce rules like min bal must be 1000.

=> different integrity constraints in sql server

1 NOT NULL

2 UNIQUE

3 PRIMARY KEY

4 CHECK

5 FOREIGN KEY

6 DEFAULT

=> 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,
   -------------------------------,
   ----------------------------
 )
```

NOT NULL :-

=> not null constraint doesn't accept null values.
=> column declared with not null is called mandatory column.

example :-

```
  CREATE TABLE emp11
  (
    empno  int,
    ename  varchar(10)  NOT NULL
   )


  insert into emp11 values(100,null)  => error
  insert into emp11 values(101,'A')
```

UNIQUE :-

=> unique constraint doesn't accept duplicates.
=> a column declared with unique into that column duplicates are not allowed.

ex :-

```
    CREATE TABLE cust
    (
        cid         int,
        cname  varchar(10)  NOT NULL,
        emailid  varchar(20)  UNIQUE
    )

                 insert into cust values(10,'A','abc@gmail.com')
             insert into cust values(11,'B','abc@gmail.com')  => ERROR
             insert into cust values(12,'C',NULL)
             insert into cust values(13,'D',NULL)                  => ERROR
```

22-nov-22

PRIMARY KEY :-

=> primary key doesn't allow duplicates and nulls.

=> it is the combination of unique & not null.

=> in tables one column must be there to uniquely idetify the records and that column must be declared with primary key.

example :-

```
CREATE TABLE emp22
(
   empno   int PRIMARY KEY ,
   ename   varchar(10) not null,
   sal        money
)
```

```
INSERT INTO emp22 VALUES(1,'A',5000)
INSERT INTO emp22 VALUES(1,'B',4000)       => ERROR
INSERT INTO emp22 VALUES(NULL,'B',4000) => ERROR
```

=> primary key doesn't allow duplicates and nulls , so using empno we can uniquely identify the employees.

=> only one primary key allowed per table. If we want multiple primary keys then declare one column with primary key and other columns with unique not null.

```
CREATE TABLE cust
(
      CUSTID         INT    PRIMARY KEY,
      NAME            VARCHAR(10) NOT NULL,
      AADHARNO    NUMERIC(12) UNIQUE NOT NULL,
     PANNO            CHAR(10) UNIQUE NOT NULL
 )
```

diff b/w primary key & unique ?

	primary key	unique
1	doesn't allow nulls	allows one null
2	a table can have only one primary key	multiple columns can be declared with unique

3	clustered index is created on primary key	non clustered index is created on unique column

CHECK constraint :-

=> use check constraint when rule based on condition.

syn :- CHECK(condition)

example 1 :- sal must be min 3000

```
CREATE TABLE EMP33
(
  EMPNO INT  PRIMARY KEY,
  ENAME VARCHAR(10) NOT NULL,
  SAL       MONEY  CHECK(SAL>=3000)
 )

   INSERT INTO EMP33 VALUES(1,'A',1000)     => ERROR
   INSERT INTO EMP33 VALUES(2,'B',5000)
   INSERT INTO EMP33 VALUES(3,'C',NULL)
```

NOTE :- check constraint allows nulls

example 2 :- gender must be 'm','f' ?

```
gender   char(1)     CHECK(geder IN ('m','f'))
```

example 3 :- amt must be multiple of 100

```
amt       money   CHECK(amt%100=0)
```

example 4 :- pwd must be min 6 chars

```
pwd     varchar(10)  CHECK(LEN(pwd)>=6)
```

example 5 :- emailid must contain '@'
                          must 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.

=> To establish relationship take primary key of one table and add it to another table as foreign key and declare with references constraint.

example :-

PROJECTS

projid	pname	duration	cost	client
100	ABC	5 YEARS	300	TATA MOTORS
101	XYZ	4 YEARS	200	DBS BANK
102	KLM	3 YEARS	150	L&T

EMP

empid	ename	sal	projid	REFERENCES PROJECTS(projid)
1	A	5000	100	
2	B	3000	101	
3	C	1000	999	=> invalid
4	D	4000	100	
5	E	3000	NULL	

=> values entered in foreign key column should match with values entered in primary key 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.

```
 CREATE TABLE projects
(
   projid  INT PRIMARY KEY,
   pname VARCHAR(10) NOT NULL
 )

INSERT INTO projects VALUES(100,'ABC'),(101,'XYZ')

CREATE TABLE emp_proj
(
  empid   INT PRIMARY KEY,
  ename  VARCHAR(10) NOT NULL,
  sal        MONEY CHECK(sal>=5000),
  projid    INT  REFERENCES projects(projid)
 )

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

Relationship Types :-

----

1 one to one (1:1)
2 one to many (1:m)   (DEFAULT)
3 many to one (m:1)
4 many to many (m:n)

=> by default sql server creates one to many relationship between two tables
to establish one to one relationship then declare foreign key with
unique constraint.

example for one to one relationship :-

DEPT

DNO	DNAME
10	HR
20	IT

MGR

MGRNO	MNAME	DNO REFERENCES DEPT(DNO) UNIQUE
1	A	10
2	B	20

=> write create table script for the above example ?

23-nov-22

many to many relationship :-

CUST

CID	NAME	ADDR
1	A	HYD
2	B	BLR

PRODUCTS

PRODID	PNAME	PRICE
100	X	1000
101	K	2000

=> relationship between cust and products is many to many

=> rdbms doesn't support many to many relationship. To establish m:n relationship create 3rd table and in 3rd table take primary keys of both tables as foreign keys

SALES

CID	PRODID	QTY	AMT
1	100	1	1000
1	101	1	2000
2	100	1	1000

Question :-

ACCOUNTS

ACCNO ACTYPE BAL

Rules :-

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

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

Rules :-

1 trid should not be duplicate & null

2 ttype must be 'w' or 'd'
3 tdate must be system date
4 tamt must be multiple of 100
5 accno should match with accounts table accno

=> write create table script ?

TABLE LEVEL :-

=> if constraints are declared after declaring all columns then it is called table level

=> use table level to declare constraints from multiple or combination of columns

Declaring check constraint at table level :-

```
PRODUCTS
prodid    pname     mfd_dt          exp_dt
100       A         2022-11-23      2022-01-01   => invalid
```

Rule :-   exp_dt > mfd_dt

=> above rule is based on multiple columns so can't be declared at column level
must be declared at table level.

```
 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(),'2022-01-01') => ERROR
```

composite primary key :-

=> if combination of columns declared with primary key then it is called composite
primary key.

=> in some tables combination of columns uniquely identifies the records so that combination should be declared as primary key at table level.

example :-

STUDENT

SID	SNAME
1	A
2	B

COURSE

CID	NAME
10	.NET
11	SQL

REGISTRATIONS

SID	CID	DOR	FEE
1	10	??	??
1	11	??	??
2	10	??	??

=> in the above example sid,cid combination uniquely identifies the records so declare this combination as primary key at table level.

```
  CREATE TABLE registrations
  (
      sid   int ,
      cid   int ,
      dor  date,
     fee   money,
                primary key(sid,cid)
 )
```

```
INSERT INTO registrations VALUES(1,10,GETDATE(),1000)
INSERT INTO registrations VALUES(1,11,GETDATE(),1000)
INSERT INTO registrations VALUES(2,10,GETDATE(),1000)
INSERT INTO registrations VALUES(1,10,GETDATE(),1000)  => ERROR
```

NOTE :-

=> all constraints can be declared at table level except NOT NULL.

Droping constraints :-

```
ALTER TABLE <tabname>
      DROP CONSTRAINT <name>
```

=> drop check constraint in emp_proj table ?

```
ALTER TABLE emp_proj
      DROP CONSTRAINT CK__emp_proj__sal__4222D4EF
```

=> drop primary key in projects table ?

```
ALTER TABLE projects
      DROP CONSTRAINT  PK__projects__3E19AD3AC312232A => ERROR

DROP TABLE projects  => ERROR

TRUNCATE TABLE projects => ERROR
```

NOTE :-

pk cannot be dropped if referenced by some fk
pk table cannot be dropped if referenced by some fk
pk table cannot be truncated if referenced by some fk

24-nov-22

Delete rules :-
----

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

=> the above rules are declared with foreign key.

ON DELETE NO ACTION :-

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

```
CREATE TABLE dept88
(
   dno  int primary key,
   dname varchar(10)
)
```

```
 INSERT INTO dept88 VALUES(10,'HR'),(20,'IT')

 CREATE TABLE emp88
 (
   empno INT PRIMARY KEY,
   ename VARCHAR(10),
   dno      INT  REFERENCES dept88(dno)
 )

INSERT INTO emp88 VALUES(1,'A',10),(2,'B',10)


 DELETE FROM  DEPT88 WHERE DNO = 10   => ERROR
```

scenario :-

ACCOUNTS

ACCNO	ACTYPE	BAL
100	S	10000

LOANS

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

NOTE :- 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 dept88
 (
   dno  int primary key,
   dname varchar(10)
 )

 INSERT INTO dept88 VALUES(10,'HR'),(20,'IT')

 CREATE TABLE emp88
 (
   empno INT PRIMARY KEY,
   ename VARCHAR(10),
   dno      INT  REFERENCES dept88(dno)
                    ON DELETE CASCADE
 )
```

```
INSERT INTO emp88 VALUES(1,'A',10),(2,'B',10)
```

```
DELETE FROM DEPT88 WHERE DNO=10 => 1 ROW AFFECTED
```

```
SELECT * FROM EMP88  => NO ROWS
```

scenario :-

ACCOUNTS

ACCNO	ACTYPE	BAL
100	S	10000

TRANSACTIONS

TRID	TTYPE	TDATE	TAMT	ACCNO
1	W	??	2000	100
2	D	??	1000	100

NOTE :- when account is closed along with account delete transactions also.

ON DELETE SET NULL :-

=> if parent row is deleted then it is deleted without deleting child rows but fk will be set to null .

```
 CREATE TABLE dept88
 (
   dno  int primary key,
   dname varchar(10)
 )
```

```
 INSERT INTO dept88 VALUES(10,'HR'),(20,'IT')
```

```
 CREATE TABLE emp88
 (
   empno INT PRIMARY KEY,
   ename VARCHAR(10),
   dno      INT  REFERENCES dept88(dno)
                 ON DELETE SET NULL
 )
```

```
INSERT INTO emp88 VALUES(1,'A',10),(2,'B',10)
```

scenario :-

PROJECTS

projid	pname	duration
100		
101		

EMP

empid	ename	sal	projid
1			100
2			101

rule :- when project is completed set the employee projid to null

summary :-

=> importance of constraints
=> types of constraints
=> declaring constraints
        column level
        table level
=>droping constraints
=> delete rules

==

JOINS

=> join is an operation performed to fetch data from two or more tables.

=> in databases related data stored in multiple tables , to gather or to combine data stored in multiple tables we need to join those tables.

Types of joins :-
----

1 inner join / equi join
2 outer join
  left join
  right join
  full join
3 non equi join
4 self join
5 cross join or cartesian join

inner join :-

=> to perform inner join between the two 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.

```
SELECT  columns
FROM tab1  INNER JOIN tab2
    ON  join condition
```

join condition :-

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

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

example :-

EMP

EMPNO	ENAME	DEPTNO
1	A	10
2	B	20
3	C	30
4	D	20
5	E	10

DEPT

DEPTNO	DNAME	LOC
10	ACCTS	NY
20	RESEARCH	
30	SALES	
40	OPERATIONS	

```
=>  display     ENAME     DNAME   ?
                -----------    ------------
```

```
                    EMP        DEPT

  SELECT ENAME,DNAME
    FROM EMP inner join DEPT
        ON  EMP.DEPTNO = DEPT.DEPTNO

   A    ACCTS
  B    RESEARCH
  C    SALES
  D    RESEARCH
  E    ACCTS
```

=> display   ENAME DEPTNO   DNAME ?

```
    SELECT ENAME,DEPTNO,DNAME
     FROM EMP inner join dept
         ON  EMP.DEPTNO = DEPT.DEPTNO   => ERROR
```

28-nov-22

NOTE :- in join queries declare table alias and prefix column names with table alias for two reasons

1 to avoid ambiguity
2 for faster execution

```
 SELECT E.ENAME,D.DEPTNO,D.DNAME
   FROM EMP AS E INNER JOIN DEPT AS D
       ON E.DEPTNO = D.DEPTNO
```

=> display employees working at NEW YORK loc ?

```
   SELECT  E.ENAME,D.DNAME,D.LOC
      FROM EMP E INNER JOIN DEPT D
          ON E.DEPTNO = D.DEPTNO  /* join condition */
    WHERE D.LOC = 'NEW YORK'      /* filter condition */
```

joining more than 2 tables :-

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

example :-

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

```
=> display     ENAME      DNAME     CITY     STATE       COUNTRY   ?
              -----------     -----------    ---------------------      -----------------
                 EMP        DEPT      LOCATIONS          COUNTRIES
```

```
 SELECT E.ENAME,
               D.DNAME,
               L.CITY,L.STATE,
               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
```

outer join :-

=> inner join returns only matching records but won't return unmatched records.
To display unmatched records also perform outer join.

example :-

EMP			DEPT		
EMPNO	ENAME	DEPTNO	DEPTNO	DNAME	LOC
1	A	10	10	ACCTS	NY
2	B	20	20	RESEARCH	
3	C	30	30	SALES	
4	D	20	40	OPERATIONS	=> unmatched record
5	E	NULL => unmatched record			

=> outer join is 3 types

1 left join

2 right join
3 full join

Left Join :-

=> returns all rows 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
```

=> returns all rows (matched + unmatched) from emp and matching rows from dept.

```
A ACCTS
B RESEARCH
C    SALES
D    RESEARCH
E    NULL          => unmatched 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
```

=> returns all rows from dept and matching from emp

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

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 ACCTS
      B RESEARCH
      C     SALES
      D     RESEARCH
      E     NULL         => unmatched from emp
    NULL OPERATIONS => unmatched from dept

   Displaying unmatched records :-
   --------------------------------------------

  left table :-
  ---------------

     SELECT E.ENAME,D.DNAME
        FROM EMP E LEFT JOIN DEPT D
            ON E.DEPTNO = D.DEPTNO
          WHERE D.DNAME IS NULL

             E       NULL

  right table :-
  ------------------

    SELECT E.ENAME,D.DNAME
      FROM EMP E RIGHT JOIN DEPT D
          ON E.DEPTNO = D.DEPTNO
     WHERE E.ENAME IS NULL

        NULL    OPERATIONS

  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
             DNAME IS NULL
```

E	NULL
NULL	OPERATIONS

Assignment :-

PROJECTS

projid	pname	duration
100		
101		
102		

EMP

empid	ename	sal	projid
1			100
2			101
3			NULL

=> display employee details with project details ?

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

=> display only the projects where no employee assigned to it ?

29-nov-22

Non Equi Join :-

=> Non Equi Join is performed between two tables not sharing a common field

ex :-

EMP

EMPNO	ENAME	SAL
1	A	5000
2	B	3000
3	C	2000
4	D	1500
5	E	1000

SALGRADE

GRADE	LOSAL	HISAL
1	700	1000
2	1001	2000
3	2001	3000
4	3001	4000
5	4001	9999

=> display ENAME SAL GRADE ?

```
                 -------------------     ------------
                   EMP                  SALGRADE
```

```
      SELECT E.ENAME,E.SAL,S.GRADE
        FROM  EMP E JOIN SALGRADE S
           ON  E.SAL  BETWEEN S.LOSAL  AND  S.HISAL
```

```
        A      5000    5
       B      3000     3
       C      2000     2
       D      1500     2
       E      1000     1
```

=> display grade 3 employee list ?

```
        SELECT 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
```

ON E.SAL BETWEEN S.LOSAL AND S.HISAL :-

EMP

EMPNO	ENAME	DEPTNO	SAL
1	A	10	5000
2	B	20	3000
3	C	30	2000
4	D	10	1500
5	E	20	1000

SALGRADE

GRADE	LOSAL	HISAL
1	700	1000
2	1001	2000
3	2001	3000
4	3001	4000
5	4001	9999

```
output :-                                              DEPT
1          A           10          5000      5          DEPTNO  DNAME  LOC
```

2	B	20	3000	3	10	ACCTS
3	C	30	2000	2	20	RESEARCH
4	D	10	1500	2	30	SALES
5	E	20	1000	1	40	OPERATIONS

ON E.DEPTNO = D.DEPTNO :-

1	A	10	5000	5	10	ACCTS
2	B	20	3000	3	20	RESEARCH
3	C	30	2000	2	30	SALES
4	D	10	1500	2	10	ACCTS
5	E	20	1000	1	20	RESEARCH

SELECT :-

A	ACCTS	5
B	RESEARCH	3
C	SALES	2
D	ACCTS	2
E	RESEARCH	1

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.
=> to perform self join the same table must be declared two times with different alias in FROM clause

FROM emp x JOIN emp y

EMP X			EMP Y		
EMPNO	ENAME	MGR	EMPNO	ENAME	MGR
1	A	NULL	1	A	NULL
2	B	1	2	B	1
3	C	1	3	C	1
4	D	2	4	D	2
5	E	3	5	E	3

=> display ENAME MGRNAME ?

```
    SELECT  X.ENAME,Y.ENAME
      FROM  EMP X JOIN EMP Y
         ON  X.MGR = Y.EMPNO

            B          A
            C          A
            D          B
            E          C
```

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

30-nov-22

TEAMS

ID	COUNTRY
1	IND
2	AUS
3	NZ

=> write a query to display following output ?

```
IND  VS  AUS
IND  VS  NZ
```

AUS VS NZ

TEAMS A		TEAMS B	
ID	COUNTRY	ID	COUNTRY
1	IND	1	IND
2	AUS	2	AUS
3	NZ	3	NZ

A.ID <> B.ID	A.ID = B.ID	A.ID > B.ID	A.ID < B.ID
IND AUS	IND IND	AUS IND	IND AUS
IND NZ	AUS AUS	NZ IND	IND NZ
AUS IND	NZ NZ	NZ AUS	AUS NZ
AUS NZ			
NZ IND			
NZ AUS			

```
 SELECT A.COUNTRY + '  VS  ' + B.COUNTRY
    FROM  TEAMS A JOIN TEAMS B
         ON A.ID < B.ID
```

CROSS JOIN / CARTESIAN JOIN :-

=> cross join returns cross product or cartesian product of two tables.

A = 1,2
B = 3,4

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

=> if cross join performed between two tables then all records of 1st table joined with all records of 2nd table.

=> to perform cross join write the join query without join condition.

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

GROUP BY & JOIN :-

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

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

FROM :-

EMP

EMPNO	ENAME	DEPTNO
1	A	10
2	B	20
3	C	30
4	D	20
5	E	10

DEPT

DEPTNO	DNAME	LOC
10	ACCTS	NY
20	RESEARCH	
30	SALES	
40	OPERATIONS	

ON e.deptno = d.deptno :-

1	A	10	ACCTS
2	B	20	RESEARCH
3	C	30	SALES
4	D	20	RESEARCH
5	E	10	ACCTS

GROUP BY d.dname :-

```
 ACCTS
             1    A   5000
             5    E   3000

 RESEARCH
            2    B   4000
            4    D   3000

  SALES
            3    C   4000
```

SELECT d.dname,SUM(e.sal) as totsal :-

```
    ACCTS      8000
    RESEARCH         7000
    SALES                4000
```

Assignment :-

SALES

DATEID	PRODID	CUSTID	QTY	AMT
2022-11-30	100	10	1	3000

PRODUCTS

PRODID	PNAME	PRICE	CATEGORY
100	AAA	3000	ELECTRONICS

CUST

CUSTID	NAME	ADDR	COUNTRY
10	KK	HYD	IND

=> display category wise total amount ?

=> display country wise total amount ?

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

===

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

=> in SQL SERVER set operations performed between two query outputs (set of rows)

```
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

UNION :-

=> combines rows
=> eliminates duplicates
=> sorts result

```
 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

SELECT JOB FROM EMP WHERE DEPTNO = 20
UNION
SELECT JOB FROM EMP WHERE DEPTNO = 30

ANALYST
```

CLERK
MANAGER
SALESMAN

UNION VS JOIN :-

	UNION	JOIN
1	combines rows	combines columns
2	horizontal merge	vertical merge
3	performed between two similar structures	can be performed between two dissimilar structures

T1	T2
F1	C1
1	10
2	20

```
T1 UNION T2 :-              T1  JOIN T2 :-
--------------------        -----------------

1                           1   10
2                           2   20
10
20
```

scenario :-

EMP_US
ENO ENAME   SAL   DNO

EMP_IND
ENO ENAME SAL   DNO

DEPT
DNO   DNAME   LOC

=> total employees list ?

SELECT * FROM EMP_US

```
UNION
SELECT * FROM EMP_IND ;
```

=> employees working at US loc with dept details ?

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

=> total employees with dept details ?

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

01-DEC-22

UNION ALL :-

=> combines rows
=> 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	duplicates are not eliminated
2	result is sorted	result is not sorted
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
```

Question :-

T1	T2
F1	C1
1	1

2	2
3	3
10	40
20	50
30	60

=> write the outputs for the following operations ?

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

SUBQUERIES / NESTED QUERIES :-

=> a query in another query is called subquery 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 then it executes outer query.
=> output of inner query is input to outer query.
=> use subquery when where cond is based on unknown value.

Types of subqueries :-

1 single row subqueries
2 multi row subqueries
3 co-related subqueries
4 derived tables and CTEs
5 scalar subqueries

single row subqueries :-

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

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

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

examples :-

=> display employees earning more than blake ?

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

=> employees who are senior to king ?

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

=> employee name earning max salary ?

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

 2    SELECT ename,max(sal)
      FROM emp                        => ERROR

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

=> name of the employee having max experience ?

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

=> display 2nd max salary ?

```
  SELECT MAX(sal)
  FROM emp
  WHERE sal <> (SELECT MAX(sal) FROM emp)
```

=> name of the employee earning 2nd max salary ?

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

=> delete the employee having max experience ?

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

=> swap employee salaries whose empno = 7499,7521 ?

```
before swap                after swap

7499    1600               7499  1250
7521    1250               7521  1600
```

```
UPDATE emp
SET sal =  CASE  empno
             WHEN 7499 THEN (SELECT sal FROM emp WHERE empno=7521)
             WHEN 7521 THEN (SELECT sal FROM emp WHERE empno=7499)
             END
WHERE empno IN (7499,7521)
```

02-dec-22

Multirow subqueries :-

=> if inner query returns more than one value then subquery is called multirow subquery.

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

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

=> list of employees working at NEW YORK,CHICAGO locations ?

```
 SELECT *
 FROM emp
WHERE deptno IN (SELECT deptno
                    FROM dept
                    WHERE loc IN ('NEW YORK','CHICAGO'))
```

ANY operator :-

=> use ANY operator for   >   <   comparision with multiple values

```
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
         1500     TRUE
         4500     FALSE
```

ALL operator :-

=>   use ALL operator fro > < comparision with multiple values

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

if x = 800          false
         1500       false
         4500       true

WHERE X < ALL(1000,2000,3000)

    if x = 800   true
           1500  false
           4500  false
```

=> employees earning more than all managers ?

```
SELECT *
FROM emp
WHERE sal >  ALL (SELECT SAL FROM EMP WHERE JOB='MANAGER')

                           2975
                           2850
                           2450
```

=> above query selects all the employees earning more than 2975

=> employees earning more than any manager ?

```
SELECT *
FROM emp
WHERE sal >  ANY (SELECT SAL FROM EMP WHERE JOB='MANAGER')
```

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

CO-RELATED subqueries :-

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

=> execution starts from outer query and inner query is executed no of times depends on no of rows return by outer query.

=> use co-related subquery to execute subquery for each row

steps :-

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

example :-

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	10
2	B	3000	20
3	C	4000	30
4	D	6000	20
5	E	3000	10

=> employees earning more than avg(sal) of their dept ?

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

```
1    A    5000 10    5000 >  (where deptno = 10)   4000   TRUE
2    B    3000  20   3000  >  (where deptno =  20)  4500   FALSE
3    C    4000 30    4000  >   (where deptno =  30)   4000   FALSE
4    D    6000 20    6000  >   (where deptno = 20)    4500   TRUE
5    E    3000 10    3000  >   ( where deptno = 10)   4000    FALSE
```

=> employees earning maximum salary in their dept ?

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

```
1    A    5000 10     5000 = (where deptno=10)  5000   TRUE
2    B    3000  20    3000 = (where deptno=20)  6000   FALSE
3    C    4000 30     4000 = (where deptno=30)  4000  TRUE
```

03-dec-22

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,

=> use derived tables in following scenarios

1 to control order of execution of clauses
2 to use result of one operation in another operation
3 to join query outputs

controlling order of execution of clauses :-

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

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

=> use derived tables to control this order of execution

example 1 :-

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

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

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

```
 SELECT ename,sal,
        dense_rank() over (order by sal desc) as rnk
FROM emp
WHERE  rnk <= 5    => ERROR
```

=> column alias cannot be used in where clause because where clause is executed before select. To control this use derived tables

```
SELECT *
FROM (SELECT ename,sal,
              dense_rank() over  (order by sal desc) as rnk
            FROM emp) AS E
WHERE rnk <= 5
```

```
SELECT *
FROM E
WHERE rnk<=5
```

=> display top 5 max salaries ?

```
SELECT DISTINCT sal
FROM (SELECT ename,sal,
              dense_rank() over  (order by sal desc) as rnk
            FROM emp) AS E
WHERE rnk <= 5
ORDER BY sal DESC
```

example 2 :-

=> display first 5 rows from emp table ?

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

```
WHERE rno=5
```

```
WHERE rno IN (5,7,10)
```

```
WHERE rno BETWEEN 5 AND 10
```

```
WHERE rno%2 = 0
```

=> display last 3 rows ?

```
 SELECT *
 FROM ( SELECT empno,ename,sal ,
                    row_number() over (order by empno asc) as rno
           FROM emp ) AS E
WHERE rno >= (SELECT COUNT(*)-2 FROM emp)
```

=> delete first 3 rows from emp table ?

```
  DELETE
  FROM ( SELECT empno,ename,sal ,
                    row_number() over (order by empno asc) as rno
           FROM emp ) AS E
 WHERE rno <= 3    => ERROR
```

in derived tables outer query cannot be DML and it must be always SELECT. To overcome this problem use CTEs.

CTE :-

=> CTE stands for common table expression , it is a named query output and we can refer this name in another queries like INSERT/UPDATE/DELETE/SELECT.

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

    SELECT/INSERT/UPDATE/DELETE
```

=> in derived tables outer query cannot be DML and it must be always SELECT but in CTEs outer query can be DML command.

example 1 :-   delete first 5 rows from emp ?

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

example 2 :- delete duplicate records from table ?

EMP33

ENO	ENAME	SAL
1	A	5000
2	B	6000
1	A	5000
2	B	6000
3	C	7000

step 1 :- generate row_numbers with in group of same eno,ename,sal

```
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
```

step 2 :- 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
```

SCALAR SUBQUERIES :-

=> subqueries in select clause are called scalar subqueries

```
    SELECT (select stmt) ,(select stmt),-----------
    FROM tabname
    WHERE cond
```

=> subquery output acts like a column for outer query
=> use scalar subquery to show the query output in seperate column

example 1 :-

```
SELECT (SELECT COUNT(*) FROM EMP) AS EMP,
       (SELECT COUNT(*) fROM DEPT) AS DEPT

   EMP      DEPT
    9        4
```

example 2 :-

=> display dept wise total salary ?

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

10    8750
20   10875
30   9400
```

=> 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     29025
20   10875     29025
30    9400     29025
```

=> 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
```

```
10      8750.00         29025.00        30.146400
20      10875.00        29025.00        37.467700
30      9400.00         29025.00        32.385800
```

SELECT stmt
        where
        order by
        distinct
        top
        functions
        group by
        joins
        set operators
        subqueries

==
==

PIVOT operator :-

=> used to convert rows into columns.
=> used for cross tabulation.
=> used to display data in matrix form.

example :-

```
                    10       20        30

   analyst          ??       ??        ??

   clerk            ??       ??        ??

  manager           ??       ??        ??

  salesman          ??       ??        ??
```

syntax :-

SELECT columns

```
  FROM (SELECT required data) AS <ALIAS>
  PIVOT
       (AGGR-EXPR FOR COLNAME IN (V1,V2,V3,---))  AS <NAME>
  ORDER BY COLNAME
```

```
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(Q,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
```

creating new table from existing table :-

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

example 1 :- (copying all columns & rows)

```
SELECT * INTO EMP10
FROM EMP
```

=> after executing above command sql server creates a new table with name EMP10
and copies structure & data from emp to emp10 .

example 2 :- (copy specific rows & cols)

```
  SELECT empno,ename,sal,job INTO emp12
  FROM emp
 WHERE  job IN ('clerk','manager')
```

example 3 :- (copy only structure (cols) but not data (rows))

```
SELECT * INTO EMP13
FROM emp
WHERE 1=2
```

example 4 :- copying table from one db to another db

=> copy db4pm students table to db7am ?

```
SELECT * INTO DB7AM.DBO.STUDENTS
FROM DB4PM.DBO.STUDENTS
```

06-dec-22

MERGE :-

=> command used to merge data into a table.
=> it is the combination of insert,update,delete.
=> used to manage replicas.
=> widely used in ETL applications.

syntax :-

```
MERGE INTO <TARGET-TABLE> <ALIAS>
USING <SOURCE-TABLE> <ALIAS>
```

```
ON (CONDITION)
WHEN MATCHED THEN
   UPDATE
WHEN NOT MATCHED THEN
   INSERT
WHEN NOT MATCHED BY SOURCE THEN
  DELETE
```

example :-

STEP 1 :- create source table

CUSTS

CID	NAME	ADDR
1	A	HYD
2	B	MUM

STEP 2 :- create target table (replica)

```
SELECT * INTO CUSTT FROM CUSTS
```

CUSTT

CID	NAME	ADDR
1	A	HYD
2	B	MUM

STEP 3 :- change the source table

```
INSERT INTO CUSTS VALUES(3,'C','DEL');

UPDATE CUSTS SET ADDR='BLR' WHERE CID = 1 ;
```

CUSTS

CID	NAME	ADDR	
1	A	HYD	=> updated
2	B	MUM	
3	C	DEL	=> inserted

STEP 4 :- replicate changes to target table using merge command

```
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.NAME,S.ADDR)
  WHEN NOT MATCHED BY SOURCE THEN
     DELETE  ;
```

Question :-

EMPS

EMPID	ENAME	SAL
1	A	5000
2	B	6000
3	C	4000

EMPT

EMPID	ENAME	SAL
1	A	
2	B	
3	C	

copy salaries from emps to empt ?

==

DB 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

SERVER (LOGINS)
    DATABASE (USERS)

==

07-DEC-22

DB objects :-

TABLES
VIEWS
SYNONYMS
SEQUENCES
INDEXES

STORED PROCEDURES
STORED FUNCTIONS
TRIGGERS

VIEWS :-

=> a view is a subset of a table.

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

Droping view :-

```
DROP VIEW V1
```

if i drop table what about views created on table ?

ans :- views are not dropped but views cannot be queried

synonyms :-

=> a synonym is another name or alternative name for a table or view.

=> if tablename is lengthy then developer can give a simple and shortname
to the table called synonym and instead of using tablename developer can use
synonym name.

syn :- CREATE SYNONYM <NAME> FOR <TABNAME>

ex :- CREATE SYNONYM E FOR EMP

=> after creating , instead of using tablename use synonym name in SELECT/INSERT/UPDATE/DELETE queries.

```
  1     SELECT * FROM E

   2    UPDATE E SET COMM=500 WHERE EMPNO = 7369
```

=> list of synonyms created by user ?

```
SELECT * FROM SYS.synonyms
```

Droping synonym :-

```
DROP SYNONYM E
```

Question :-

```
1       CREATE SYNONYM E FOR EMP

2       SELECT * FROM EMP AS E

3       SP_RENAME 'EMP','E'   =>  changes tablename from emp to e
```

=> diff b/w synonym and alias ?

	synonym	alias
1	permanent	temporary
2	stored in db	not stored in db
3	scope of the synonym is upto the schema	scope of the alias is upto the query

SEQUENCES :-

=> sequence is also a db object created to generate sequence numbers
=> used to auto increment column values.

syn :-

```
 CREATE SEQUENCE <NAME>
 [START WITH <value>]
 [INCREMENT BY <value>]
 [MAXVALUE <value>]
 [MINVALUE <value>]
 [CYCLE/NOCYCLE]
 [CACHE <size>]
```

Ex :-

```
 CREATE SEQUENCE S1
 START WITH 1
 INCREMENT BY 1
 MAXVALUE 5
```

using sequence :-

```
 CREATE TABLE stud
 (
   sid   int,
   sname varchar(10)
 )

 INSERT INTO stud VALUES(NEXT VALUE FOR S1, 'A')
 INSERT INTO stud VALUES(NEXT VALUE FOR S1, 'B')
 INSERT INTO stud VALUES(NEXT VALUE FOR S1, 'C')
 INSERT INTO stud VALUES(NEXT VALUE FOR S1, 'D')
 INSERT INTO stud VALUES(NEXT VALUE FOR S1, 'E')
 INSERT INTO stud VALUES(NEXT VALUE FOR S1, 'F')   => ERROR

 SELECT * FROM STUD

 SID    NAME
1       A
2       B
3       C
4       D
5       E
```

example 2 :- calling sequence in update command

```
 CREATE SEQUENCE S2
START WITH 100
INCREMENT BY 1
MAXVALUE 999
```

=> use above sequence to update empno

```
UPDATE EMP SET EMPNO = NEXT VALUE FOR S2
```

cycle/nocycle :-

=> by default sequence created with nocycle.

=> if nocycle then it starts from start with and generates upto max and after reaching max then it stops .

=> if cycle then it starts from start with and generates upto max and after reaching max then it will be reset to min.

Ex :-

```
create sequence s3
start with 1
increment by 1
maxvalue 5
minvalue 1
cycle
```

CACHE 100 :-

```
CREATE SEQUENCE S4
START WITH 1
INCREMENT BY 1
MAXVALUE 1000
MINVALUE 1
CYCLE
CACHE 100
```

=> sql server preallocates 100 values in cache memory , so everytime we call sequence next value then it goes to db and gets the values and returns that value , accessing cache memory is much faster than accessing db. so this improves performance.

=> list of sequenes created by user ?

```
select * from INFORMATION_SCHEMA.sequences
```

Droping sequence :-

```
DROP SEQUENCE S1
```

IDENTITY :-

=> used to auto increment column values

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

  SEED => start
          default 1

 INCR  =>  increment
           default 1
```

example :-

```
CREATE TABLE cust
(
  CID   INT     IDENTITY(100,1),
  NAME VARCHAR(10)
)

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

SELECT * FROM cust

cid     name
100   A
101   B
102   C
```

103   D

10-dec-22

=> diff b/w identity & sequence ?

	identity	sequence
1	bind to a column	not bind to any column
2	cannot be declared with maxvalue	can be declared with max value
3	identity cannot be reset	can be reset
4	identity cannot be accessed from application programs	can be accessed from application program

===

INDEXES :-

=> indexes are created to improve the performance of data accessing.

=> index improves the performance of search operation i.e. searching for records.

=> index in db is similar to index in textbook , in textbook using index a particular topic can be located fastly , in db using index a particular record can be located fastly.

=> indexes are created on columns and that column is called index key.

=> indexes are created on

1 columns that are frequently used in where clause
2 columns that are used in join operation

Types of indexes :-
----

1 Non clustered Indexes

- simple
- composite
- unique

2 Clustered Indexes

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)

```
                                        index

EMP                                      3000
SAL
2000
5000                   2000                           4000
3000
1500            1000 *          2500 *          4000 *          5000 *
4000            1500 *           3000 *,*
2500            2000 *
1000
3000
```

=> when we submit a query to sql server ,it uses following methods to locate the row

1 table scan
2 index scan

=> in table scan sql server scans complete table i.e. each and every row.

=> in index scan on avg sql server scans only half of the table , so index scan is much faster than table scan.

```
SELECT * FROM emp WHERE sal = 3000 ;   (INDEX SCAN)
SELECT * FROM emp WHERE sal>=3000 ;   (INDEX SCAN)
SELECT * FROM emp WHERE sal<= 3000;   (INDEX SCAN)

SELECT * FROM EMP                        (TABLE SCAN)
SELECT * FROM EMP WHERE ENAME='BLAKE'   (TABLE SCAN)
SELECT * FROM EMP WHERE SAL <:> 3000 ;       (TABLE SCAN)
```

composite index :-

=> if index created on multiple columns then index is called composite index

```
CREATE INDEX I2 ON EMP(DEPTNO,JOB)
```

unique index :-

=> unique index doesn't allow duplicate values into the column on which index is created

```
CREATE UNIQUE INDEX I3 ON EMP(ENAME)
```

```
                                    K

                     G                                  Q

  ADAMS *               JAMES *        MARTIN *       SCOTT *
  ALLEN *               JONES *         MILLER *       SMITH *
 BLAKE *
```

```
SELECT * FROM EMP WHERE ENAME='BLAKE'  ;    (index scan)

INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(888,'BLAKE',4000) => ERROR
```

=> what are the different methods to enforce uniqueness ?

ans :-

1 primary key / unique constraint
2 create unique index

=> primary key / unique columns are automatically indexed by sql server and sql server creates unique index on primary key / unique columns and unique index doesn't allow duplicates so primary key / unique also doesn't allow duplicates

12-dec-22

clustered indexes :-

=> a Non clustered index stores pointers to actual records where as clustered index stores actual records

```
ex :-  create table cust
        (
           cid   int,
           cname varchar(10)
        )

       create clustered index i10 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(70,'D')
```

```
                                    60

                       30                         70

              10   A         40   C        70 D         80  B
```

```
   SELECT * FROM cust  => sql server goes to clustered index and access
                          all the leaf nodes from left to right
    10   A
    40   C
    70   D
    80   B

  SELECT * FROM cust WHERE cid = 40
```

NOTE :-

1   only one clustered index is allowed per table.

2   sql server implicitly creates clustered index on primary key column column

diff b/w clustered and non clustered indexes ?

	clustered	non clustered
1	stores actual records	stores pointers to actual records
2	order of elements in index and table is same	order of elements in index and table is not same
3	doesn't need extra storage	needs extra storage
4	requires only one lookup to access record	requires two lookups to access records
5	only one clustered index allowed per table	999 non clustered indexes allowed per table
6	implicitly created on primary key column	implicitly created on unique columns

=> how to see the list of indexes created on emp table ?

```
sp_helpindex  emp
```

Droping indexes :-

```
DROP INDEX EMP.I1
```

=> if we drop table what about indexes created on table ?

ans :- indexes are also dropped

```
SERVER
      DATABASE
              TABLES
                      ROWS & COLS
                      CONSTRAINTS
                      INDEXES
                      TRIGGERS
              VIEWS
              SYNONYMS
              SEQUENCES
```

==

SQL

commands	clauses		operators	objects
DDL	where	=> data filtering	between	tables
DML	order by	=> sorting	in	views
DQL	distinct	=> eliminating duplicates	like	synonyms
TCL	top	=> top N rows	is	sequences
DCL	group by	=> grouping	any	indexes
	having	=> filter after group by	all	
	on	=> join	exists	stored proc
	with	=> cte	pivot	functions
				triggers

T-SQL programming :- (Transact-SQL)

Basic programming
conditional stmts
loops
cursors
error handling
stored procedures
functions
triggers
dynamic sql

Features :-
----

1 improves performance :-

=> in TSQL , sql commands can be grouped into one block and we submit that block to sql server , so in TSQL no of requests and response between user and sql server are reduced and performance is improved.

2 supports conditional statements :-

=> supports conditional statements like IF-ELSE .

3 supports loops :-

=> tsql supports looping statements like while

4 supports error handling :-

=> in tsql , if any statement causes error then we can handle that error and we can display our own simple and user friendly message.

5 support reusability :-

=> tsql programs can be stored in db and applications which are connected to db can reuse these programs.

=> TSQL programs are 2 types

1 Anonymous Blocks
2 Named Blocks
    stored procedures
    functions
    triggers

Anonymous Blocks :-

=> a tsql program without name is called anonymous block.

=> the following statements are used in tsql programming.

1 DECLARE

2 SET
3 PRINT

13-dec-22

Declare statement :-

=> used to declare variables

```
syn :-  DECLARE  @VARNAME    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

```
syn :-      SET @varname =  value

ex  :-      SET @x = 100
            SET @s = 'abc'
            SET @d = GETDATE()
```

PRINT statement :-

=> used to print messages or values

```
PRINT @x
PRINT 'hello'
```

example 1 :-

```
DECLARE @a INT,@b INT,@c INT
SET @a=100
SET @b=200
SET @c=@a+@b
```

```
   PRINT @c
```

example 2 :- write a prog to input date and print day of the week ?

```
    DECLARE @d  DATE
    SET @d = GETDATE()
    PRINT DATENAME(DW,@d)
```

DB programming with TSQL :-

=> to perform operations over db execute SQL commands from tsql program.

=> the following commands are 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,--------
FROM tabname
WHERE condition
```

ex :-

```
SELECT @a = ename ,@b=sal  FROM emp WHERE empno = 107
```

=> write a prog to input empno and print name & salary ?

```
    DECLARE @eno INT,@name VARCHAR(10),@sal MONEY
   SET @eno=111
   SELECT  @name=ename,@sal=sal FROM emp WHERE empno = @eno
   PRINT @name + '    '  + CAST(@sal as varchar)
```

=> write a prog to input empno and print experience of the employee ?

```
    DECLARE @eno INT,@hire   DATE,@expr  INT
   SET @eno=105
```

```
    SELECT @hire=hiredate FROM emp WHERE empno = @eno
    SET @expr =  DATEDIFF(YY,@hire,GETDATE())
    PRINT 'experience = ' + CAST(@expr AS VARCHAR) + ' years'
```

conditional statements :-

----

1 IF-ELSE
2 MULTI IF
3 NESTED IF

IF-ELSE :-

```
 IF COND
   BEGIN
        statements
   END
ELSE
    BEGIN
       statements
   END
```

MULTI IF :-

```
  IF  COND1
     BEGIN
         statements
     END
ELSE IF COND2
     BEGIN
          statements
     END
ELSE IF COND3
     BEGIN
         statements
     END
ELSE
      BEGIN
         statements
      END
```

nested if :-

```
  IF  COND
     BEGIN
            IF  COND
               BEGIN
                     statements
               END
           ELSE
                BEGIN
                     statements
                 END
     END
 ELSE
      BEGIN
          statements
      END
```

=> write a prog to input empno and increment sal by specific amount
    and after increment if sal exceeds 5000 then cancel that increment ?

```
  DECLARE @eno INT,@amt MONEY ,@sal MONEY
  SET @eno = 107
  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 salary as follows ?

```
     if job=CLERK  incr sal by 10%
              SALESMAN          15%
              MANAGER            20%
              OTHERS              5%
```

```
    DECLARE @eno INT,@job  VARCHAR(10),@pct INT
    SET @eno=101
    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
```

14-dec-22

=> write a prog to process bank transactions (w/d) ?

ACCOUTS

ACCNO	ACTYPE	BAL
100	S	10000
101	S	20000

```
DECLARE @acno int,@type  char(1),@amt money,@bal money
SET @acno=100
SET @type='w'
SET @amt=100
IF @type='w'
 BEGIN
      SELECT @bal=bal FROM accounts WHERE accno = @acno
      IF @amt  > @bal
          PRINT 'insufficient balance'
      ELSE
          UPDATE accouts SET bal = bal - @amt WHERE accno=@acno
   END
 ELSE IF  @type='d'
        UPDATE accounts SET bal = bal + @amt WHERE accno = @acno
 ELSE
      PRINT 'invalid trasaction type'
```

=> write prog to process money transfer ?

while loop :-

=> loops are used to execute statements repeatedly multiple times

```
  while(condition)
  begin
       statements
  end
```

if cond = true loop continues
if cond = false loop terminates

=> write a prog to print numbers from 1 to 20 ?

```
DECLARE @x  int = 1
WHILE(@x<=20)
BEGIN
    PRINT @x
    SET @x = @x + 1
END
```

=> write a prog to print numbers from 20 to 1 ?

```
 DECLARE @x  int = 20
WHILE(@x>=1)
BEGIN
    PRINT @x
    SET @x = @x - 1
END
```

=> write a prog to print 2023 calendar ?

```
2023-01-01     ???
2023-01-02     ???

2023-12-31     ???
```

```
DECLARE @d1 DATE,@d2 DATE
SET @d1 = '2023-01-01'
SET @d2 = '2023-12-31'
WHILE(@d1<=@d2)
BEGIN
    PRINT cast(@d1 AS VARCHAR)  + '    ' + DATENAME(DW,@d1)
    SET @d1 = DATEADD(DD,1,@d1)
END
```

=> write a prog to print sundays between two given dates ?

```
DECLARE @d1 DATE,@d2 DATE
SET @d1 = '2023-01-01'
SET @d2 = '2023-12-31'
/* to find first sunday */
WHILE(DATENAME(DW,@d1)<>'sunday')
BEGIN
  SET @D1 = DATEADD(DD,1,@D1)
END
 /* to print sundays */
WHILE(@d1<=@d2)
BEGIN
     PRINT cast(@d1 AS VARCHAR)  + '    ' + DATENAME(DW,@d1)
     SET @d1 = DATEADD(DD,7,@d1)
END
```

=> write a prog to input string and print following pattern ?

input :- NARESH

output :-

```
N
A
R
E
S
H
```

```
DECLARE @s  VARCHAR(10),@x INT = 1
SET @s = 'NARESH'
WHILE(@x <= LEN(@s))
BEGIN
      PRINT  SUBSTRING(@s,@x,1)
      SET @x = @x+1
END
```

=> write a prog to print following pattern ?

input :- NARESH

output :-

```
 N
 NA
 NAR
 NARE
 NARES
 NARESH
```

```
    DECLARE @s  VARCHAR(10),@x INT = 1
    SET @s = 'NARESH'
    WHILE(@x <= LEN(@s))
    BEGIN
         PRINT  SUBSTRING(@s,1,@x)
         SET @x = @x+1
    END
```

=> write a prog to input string and print reverse of that string ?

input :- NARESH

output :- HSERAN

```
 DECLARE @s1  varchar(10),@s2  varchar(10)=' ',@b int
 SET @s1 = 'NARESH'
 SET @b = len(@s1)
 WHILE(@b>0)
 BEGIN
    SET @s2 = @s2 + SUBSTRING(@s1,@b,1)
    SET  @b=@b-1
 END
 PRINT @s2
 IF @s1 = LTRIM(@s2)
    PRINT   '  palindrome'
 ELSE
   PRINT 'not a palindrome'
```

==

15-DEC-22

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 gets 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 rercords
4 close cursor
5 deallocate cursor

Declaring cursor :-
----

syn :- DECLARE <NAME> CURSOR FOR SELECT STATEMENT

ex :- DECLARE C1 CURSOR FOR SELECT ENAME,SAL FROM EMP

Opening cursor :-

OPEN <cursor-name> ;

ex :- OPEN C1 ;

1 select stmt submitted to sql server
2 data returned by select stmt is copied to temporary memory
3 cursor c1 points to that temporary memory

Fetching records :-

=> fetch stmt is used to fetch records

syn :- FETCH NEXT FROM <cursor-name> INTO <variables> ;

ex :- FETCH NEXT FROM C1 INTO @a,@b

=> fetch stmt fetches one row at a time but to process multiple rows fetch stmt should be in a loop.

closing cursor :-

```
close <cursor-name>

ex :-   close c1
```

Deallocate cursor :-

```
deallocate <cursor-name>

ex :-      deallocate c1
```

@@FETCH_STATUS :-

=> it is a system variable that returns

```
 0       =>  if fetch successful

-1       =>  if fetch unsuccessful
```

example 1 :-

=> 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 sal without using sum function ?

```
DECLARE C1 CURSOR FOR SELECT sal FROM emp
DECLARE @sal money,@t  money=0
OPEN C1
FETCH NEXT FROM C1 INTO @sal
WHILE(@@FETCH_STATUS=0)
BEGIN
    SET @t = @t + @sal
    FETCH NEXT FROM C1 INTO @sal
END
     PRINT @t
     CLOSE C1
     DEALLOCATE C1
```

16-dec-22

=> write a prog to find max sal without using max function ?

```
DECLARE C1 CURSOR FOR SELECT sal FROM emp
DECLARE @sal money,@m money = 0
OPEN C1
FETCH NEXT FROM C1 INTO @sal
WHILE(@@FETCH_STATUS=0)
BEGIN
    IF @sal > @m
        SET @m = @sal
    FETCH NEXT FROM C1 INTO @sal
END
    PRINT @m
    CLOSE C1
    DEALLOCATE C1
```

=> write a prog to find min sal without using min function ?

=> write a prog to calculate total ,avg,result of all the students and insert into result table ?

STUDENT

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

RESULT

SNO	TOTAL	AVG	RESULT

```
DECLARE C1 CURSOR FOR SELECT sno,s1,s2,s3 FROM student
DECLARE @sno int,@s1 int,@s2 int,@s3 int
DECLARE @total int,@avg decimal(5,2),@res char(4)
OPEN C1
FETCH NEXT FROM C1 @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 @sno,@s1,@s2,@s3
END
    CLOSE C1
    DEALLOCATE C1
```

SCROLLABLE CURSOR :-

=> by default cursor is forward only cursor and it supports forward navigation but doesn't support backward navigation.

=> if cursor declared with SCROLL then it is called scrollable cursor and it supports both forward and backward navigation.

```
DECLARE C1 CURSOR SCROLL FOR SELECT STATEMENT
```

=> a 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
```

=> write a prog to print last rec to first rec ?

```
     DECLARE C1 CURSOR SCROLL FOR SELECT ename FROM emp
     DECLARE @name varchar(10)
     OPEN C1
     FETCH LAST FROM C1 INTO @name
     WHILE(@@FETCH_STATUS=0)
     BEGIN
          PRINT @name
          FETCH PRIOR FROM C1 INTO @name
    END
        CLOSE C1
        DEALLOCATE C1
```

=> write a prog to print every 5th rec in emp table ?

```
     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
```

===

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
```

=> 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 called TRY------CATCH block

```
BEGIN TRY
      statements       =>   causes runtime error
END TRY
BEGIN  CATCH
    statements         =>  handles runtime error
END CATCH
```