SQL SERVER

Database :-

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

Types of Databases :-
---

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

=> organizations uses OLTP for storing day-to-day transactions and OLAP for analysis.
=> OLTP is used for running business and OLAP is used to analyze business.
=> 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.
=> DBMS also provides features like security,transaction management and concurrency control

Evolution of DBMS :-

1960 FMS (File Management System)

1970 HDBMS (Hierarchical DBMS)
NDBMS (Network DBMS)

1980 RDBMS (Relational DBMS)

1990 ORDBMS (Object -Relational DBMS)
OODBMS (Object Oriented DBMS)

RDBMS :-

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

Example :-

CUSTOMERS
CID NAME CITY AGE => COLUMNS/FIELDS/ATTRIBUTES
10 sachin mum 45
11 vijay del 30
12 rahul hyd 28 => 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

=> one table related to antoher table using foreign key.

ORDERS
ORDID ORD_DT DEL_DT CID
1000 20- 25- 10
1001 21- 28- 11
1002 21- 29- 12

CUSTOMERS
CID NAME ADDR
10 A HYD
11 B HYD
12 C HYD

23-aug-21

RDBMS softwares :-

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

ORDBMS :- (object relational dbms)

=> ORDBMS is combination of RDBMS & OOPS

ORDBMS = RDBMS + OOPS (reusability)

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

ORDBMS softwares :-

SQL SERVER
ORACLE

summary :-

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

SQL SERVER

=> SQL SERVER is basically a RDBMS product from Microsoft which is used to create and to manage database.

=> SQL SERVER software can be used for both db development and db administration.

DEVELOPER	DBA (DATABASE ADMINISTRATOR)
creating tables	installation of sql server
creating views	creating database
creating synonyms	creating logins for developers
creating sequences	db backup & restore
creating indexes	db export & import
creating procedures	db mirroring & replication
creating functions	performance tuning
creating triggers	
writing queries	

versions of sql server :-

---

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

sql server 2016 :-

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

sql server 2017 :-

1 identity cache
2 New String functions
3 Automatic Tuning

sql server 2019 :-

1 Read, write, and process big data from Transact-SQL
2 Easily combine and analyze high-value relational data with high-volume big data.

3 Query external data sources.
4 Store big data in HDFS managed by SQL Server.
5 Query data from multiple external data sources through the cluster

CLIENT/SERVER ARCHITECTURE :-
---

1 SERVER
2 CLIENT

=> SERVER is a system where sql server software is installed running
=> inside the server SQL SERVER manages databases.
=> using client system users can

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

CLIENT TOOL :-

SSMS (SQL SERVER MANAGEMENT STUDIO)

24-AUG-21

How to connect to sql server :-

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

SERVER TYPE :- Database -Engine
SERVER NAME :- WINCTRL-F9B3VH5\SQLExpress
AUTHENTICATION :- SQL SERVER authentication
LOGIN :- SA (SYSTEM ADMIN)
PASSWORD :- 123

CREATING DATABASE IN SQL SERVER :-

=> to create new databse in Object -Explorer select -Databases => New -Database

Enter Database Name :- DB4PM

=> click OK

=> a New database is created with following two files

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

=> Data File stores data and Log file stores operations i.e. commands executed on database.

NAME TYPE SIZE AUTO GROWTH PATH
DB4PM DATA 8MB 64MB C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL\DATA\
DB4PM_LOG LOG 8MB 64MB C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL\DATA\

TSQL (Transact-SQL)

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

SQL SERVER	ORACLE	MYSQL	POSTGRESQL	DB2
SQL	SQL	SQL	SQL	SQL

USERS----SSMS----------SQL----------SQL SERVER-------DB
tool language software storage

USERS----SQLPLUS----------SQL----------ORACLE---------DB

USERS-----MYSQL WORKBENCH-------SQL----------MYSQL---------DB

=> based on operations over db SQL is categorized into 5 sublanguages.

DDL (DATA DEFINITION LANG)
DML (DATA MANIPULATION LANG)
DQL/DRL (DATA QUERY LAN / DATA RETRIEVAL LANG)
TCL (TRANSACTION CONTROL LANG)
DCL (DATA CONTROL LANG)

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

25-aug-21

Datatypes in SQL SERVER :-

=> in SQL SERVER columns are declared with datatype for two reasons

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

DATATYPES

CHAR	INTEGER	FLOAT	CURRENCY	DATE	BINARY
ASCII	UNICODE	tinyint	decimal(p,s)	smallmoney	date
binary					

smallint money time
varbinary
char nchar int datetime
varbinary(max)
varchar nvarchar bigint
varchar(max) nvarchar(max)

CHAR(size) :-

=> a char datatype allows character data upto 8000 chars
=> recommended for fixed length char columns

ex :- NAME CHAR(10)

sachin----
wasted

ravi------
wasted

=> in char datatype extra bytes are wasted so char recommended for variable length fields and char is recommended for fixed length fields.

ex :- GENDER CHAR(1)

M
F

STATE_CODE CHAR(2)

AP
TS
MH

COUNTRY_CODE CHAR(3)

IND
USA

VARCHAR(size) :-

=> allows character data upto 8000 chars
=> recommended for variable length fields.
=> in varchar datatype extra bytes are released.

ex :- NAME VARCHAR(10)

sachin----
released

ravi------
released

VARCHAR(MAX) :-

=> allows character data upto 2GB.
=> in varchar(max) extra bytes are released.
=> fields declared with varchar(max) occupies more memory so these fields are called LOBs (LARGE OBJECTS)

NOTE :- char/varchar/varchar(max) allows ascii chars(256 chars) that includes a-z,A-Z,0-9,special chars. so these types allows alphanumeric data.

ex :- PANNO CHAR(10)
VEHNO VARCHAR(10)
EMAILID VARCHAR(30)
PWD VARCHAR(12)

NCHAR/NVARCHAR/NVARCHAR(MAX) :- (N => National)

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

=> ASCII char occupies 1 byte but UNICODE char occupies 2 bytes.

Integer Types :-

=> allows whole numbers i.e. numbers without decimal part.
=> sql server supports 4 integer types

TINYINT 1 BYTE 0 TO 255
SMALLINT 2 BYTES -32768 TO 32767
INT 4 BYTES -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)
BIGINT 8 BYTES -2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)

EX :- AGE TINYINT
EMPID SMALLINT
AADHARNO BIGINT

DECIMAL(P,S) :-

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

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

ex :- SALARY DECIMAL(7,2)

5000 => ACCEPTED
5000.50 => ACCEPTED
50000.50 => ACCEPTED
500000.50 => NOT ACCEPTED

5000.507 => ACCEPTED => 5000.51
5000.503 => ACCEPTED => 5000.50

CURRENCY TYPES :-

=> currency types are used for fields related to money
=> sql server supports 2 currency types

SMALLMONEY 4 BYTES -214748.3648 to 214748.3647
MONEY 8 BYTES -922,337,203,685,477.5808 to 922,337,203,685,477.5807

EX :- SALARY SMALLMONEY
BAL MONEY

DATE :-
---

type	Format	Range
date	YYYY-MM-DD	0001-01-01 through 9999-12-31
time	hh:mm:ss.[nnnnnnn]	00:00:00.0000000 to 23:59:59.9999999
smalldatetime	YYYY-MM-DD hh:mm:ss	1900-01-01 through 2079-06-06
datetime	YYYY-MM-DD hh:mm:ss[.nnn]	1753-01-01 through 9999-12-31
datetime2	YYYY-MM-DD hh:mm:ss[.nnnnnnn]	0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999

ex :- DOB DATE

1995-05-10

LOGIN TIME

10:00:00

TXN_DATE DATETIME

Binary types :-

=> binary types allows binary data that includes audio,video,images
=> SQL SERVER supports 3 binary types

BINARY
VARBINARY
VARBINARY(MAX)

BINARY :-

=> allows binary data upto 8000 bytes
=> recommended for fixed length fields
=> extra bytes are wasted

ex :- photo binary(1000)

VARBINARY :-

=> allows binary data upto 8000 bytes
=> recommended for variable length fields
=> extra bytes are released

ex :- photo varbinary(8000)

VARBINARY(MAX) :-

=> allows binary data upto 2GB.
=> extra bytes are released

ex :- photo varbinary(max)

26-aug-21

CREATING TABLE IN SQL SERVER DB :-

```
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 1024 columns
5 table can have unlimited rows

emp123	VALID
123emp	INVALID
emp 123	INVALID
emp*123	INVALID
emp_123	VALID

Example :-

create table with following structure ?

EMP

EMPID	ENAME	JOB	SAL	HIREDATE	AGE

```
CREATE TABLE emp
(
   empid      TINYINT,
   ename      VARCHAR(10),
   job        VARCHAR(10),
   sal        DECIMAL(7,2),
```

```
    hiredate   DATE,
    age              TINYINT
)
```

=> above command created table structure that includes columns,datatype and size

SP_HELP :- (SP => stored procedure)

=> command to see the structure of the table

```
        SP_HELP emp

    colname          datatype        size

    empid        tinyint        1
ename        varchar      10
job      varchar  10
sal      decimal      5
hiredate date     3
age      tinyint        1
```

INSERTING DATA INTO TABLE :-

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

1 single row
2 multiple rows

INSERTING SINGLE ROW :-

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

Ex :-

```
INSERT INTO emp VALUES(100,'sachin','clerk',4000,'2020-10-15',45)
INSERT INTO emp VALUES(101,'ravi','analyst',8000,GETDATE(),30)
```

INSERTING MULTIPLE ROWS :-

```
INSERT INTO emp VALUES(102,'kumar','manager',10000,'2019-05-10',28),
                      (103,'david','clerk',5000,GETDATE(),35)
```

INSERTING NULLS :-

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

method 1 :-

```
INSERT INTO emp VALUES(104,'satish',NULL,NULL,GETDATE(),25)
```

method 2 :-

INSERT INTO emp(empid,ename,hiredate,age) VALUES(105,'phani',GETDATE(),32)

remaining two fields job,sal filled with nulls

Displaying Data :-

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

syn :- SELECT columns/* FROM tabname

SQL = ENGLISH
QUERIES = SENTENCES
CLAUSES = WORDS

=> display employee names and salaries ?

SELECT ename,sal FROM emp

=> display employee names,salaries and jobs ?

SELECT ename,sal,job FROM emp

=> display all the data from emp ?

SELECT * FROM emp

* => all columns

Operators in SQL SERVER :-

Arithmetic Operators => + - * / %
Relational Operators => > >= < <= = <> or !=
Logical Operators => AND OR NOT
Special Operators => BETWEEN
IN
LIKE
IS
ANY
ALL
EXISTS
PIVOT & UNPIVOT
Set Operators => UNION
UNION ALL
INTERSECT
EXCEPT

27-aug-21

WHERE clause :-

=> used to get specific row/rows from table based on a condition
=> where clause is always associated with condition

```
SELECT columns/*
FROM   tabname
WHERE condition
```

condition :-

```
COLNAME   OP   VALUE
```

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

=> if cond = true row is selected , if cond = false row is not selected

=> display employee details whose empid=103 ?

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

=> display employee details whose name = satish ?

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

=> display employees earning more than 5000 ?

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

=> display employees age less than 30 ?

```
SELECT * FROM emp WHERE age < 30
```

=> display employees joined after 2020 ?

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

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

=> display employees 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	RESULT
	T	T	T	
	T		F	F
	F		T	F
	F		F	F

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

F F F

=> display employees working as clerk,manager ?

SELECT * FROM emp WHERE job='clerk','manager' => ERROR

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

=> display employees working as working as clerk and earning more than 4000 ?

SELECT * FROM emp WHERE job='clerk' AND sal>4000

=> display employees earning more than 5000 and less than 10000 ?

SELECT * FROM emp WHERE sal>5000 AND sal<10000

=> display employees joined in 2020 year ?

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

=> display employee details whose empid=100,103,105 ?

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

scenario :-

```
CREATE TABLE STUDENT
(
  SNO TINYINT,SNAME VARCHAR(10),S1 TINYINT,S2 TINYINT,S3 TINYINT
 )

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

STUDENT

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

=> display list of students who are passed ?

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

=> display list of students who are failed ?

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

IN operator :-

=> use IN operator for list comparision
=> use IN operator for "=" comparision with multiple values

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

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

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

=> display employees working as clerk,manager ?

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

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

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

=> display employees not working as clerk,manager ?

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

BETWEEN operator :-

=> use BETWEEN operator for range comparision.

WHERE COLNAME BETWEEN V1 AND V2 (WHERE COL>=V1 AND COL<=V2)
WHERE COLNAME NOT BETWEEN V1 AND V2

=> display employees earning between 5000 and 10000 ?

SELECT * FROM emp WHERE sal BETWEEN 5000 AND 10000

=> display employees joined 2020 year ?

SELECT * FROM emp WHERE hiredate BETWEEN '2020-01-01' AND '2020-12-31'

=> display employees who are not in the age group 30 to 40 ?

SELECT * FROM emp WHERE age NOT BETWEEN 30 AND 40

Question :-

SELECT * FROM emp WHERE sal BETWEEN 10000 AND 5000

A ERROR
B RETURNS ROWS
C RETURNS NO ROWS
D NONE

ANS :- C

SELECT * FROM emp WHERE sal BETWEEN 5000 AND 10000 (WHERE sal>=5000 and sal<=10000)

SELECT * FROM emp WHERE sal BETWEEN 10000 AND 5000 (WHERE sal>=10000 and sal<=5000)

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

=> DISPLAY employees working as clerk,manager and earning between 5000 and 10000

and joined in 2021 year and age between 30 and 40 ?

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

LIKE operator :-

=> use LIKE operator for pattern comparision

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

=> pattern contains alphabets,digits,wildcard characters.

wildcard chars :-

% => 0 or many chars

_ => exactly 1 char

=> display employees name starts with 's' ?

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

=> display employees name ends with 's' ?

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

=> display employees name contains 's' ?

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

=> display employees where 'a' is the 2nd char in their name ?

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

=> display employees where 'a' is the 3rd char from last ?

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

=> display employees where name contains 4 chars ?

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

=> display employees joined in jan month ?

```
YYYY-MM-DD

SELECT * FROM emp WHERE hiredate LIKE '_____01___'
```

=> display employees joined in 1981 year ?

```
SELECT * FROM emp WHERE hiredate LIKE '1981%'
```

=> display employees joined 1981 or 1983 year ?

```
SELECT * FROM emp WHERE hiredate LIKE '1981%'
                        OR
                        hiredate LIKE '1983%'
```

Question :-

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

A ERROR
B RETURNS NO ROWS
C RETURNS CLERK,MANAGER,SALESMAN
D RETURNS ONLY CLERK

ANS :- D

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

ANS :- C

Assignment 1 :-

PRODUCTS
prodid pname price category brand

1 display all the products name,price,brand ?
2 display list of mobiles phones ?
3 display mobile phones,tvs,laptops ?
4 display list of mobiles phones price between 10000 and 20000 ?
5 display list of mobile phones whose brand = realme,redmi,samsung ?
6 display list of mobile phones price between 10000 and 20000 and
brand = realme,redmi,samsung ?

Assignment 2 :-

CUSTOMERS
cid cname gender age city state

1 display list of male customers ?
2 display list of male customers living in hyd ?
3 display list of customers living in hyd,mum,del ?
4 display list of customers age between 30 and 40 ?
5 display list of male customers living in hyd,mum,del and age between 30 and 40 ?

IS operator :-

=> use IS operator for NULL comparision

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

=> display employee not earning commission ?

```
SELECT * FROM emp WHERE comm IS NULL
```

=> display employees earning commission ?

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

summary :-

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

30-AUG-21

ORDER BY clause :-

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

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

ASC => acedning
DESC => descending

=> default order is ASC

=> arrange employee list name wise asc order ?

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

31-AUG-21

=> arrange employee list sal wise desc order ?

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

=> arrange employee list hiredate wise asc order ?

```
SELECT *
FROM emp
```

```
  ORDER BY hiredate ASC
```

NOTE :- in ORDER BY we can use column names or column numbers .

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

=> above query sorts employee list based on 6th column i.e. sal

NOTE :- ORDER BY number is not based on table it should be based on SELECT list.

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

=> to sort based on sal

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

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

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

scenario :-

```
CREATE TABLE student
 (
    SNO INT,
    SNAME VARCHAR(10),
    M TINYINT,
    P TINYINT,
    C TINYINT
)

 INSERT INTO STUDENT VALUES(1,'A',80,90,70),(2,'B',60,70,50),
                           (3,'C',90,80,70),(4,'D',90,70,80)
```

STUDENT

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 student list avg wise desc order , m desc,p desc ?

```
  SELECT *
  FROM student
  ORDER BY (M+P+C)/3 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
```

=> to display avg in output execute the following ?

```
SELECT SNO,SNAME,M,P,C,(M+P+C)/3    AS AVG
FROM student
ORDER BY (M+P+C)/3 DESC,M DESC,P DESC
```

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

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

DISTINCT clause :-

=> used to select distinct values
=> eliminates duplicates from the select statement output

```
syn :- DISTINCT COL
          DISTINCT COL1,COL2,--
          DISTINCT *
```

```
SELECT DISTINCT job FROM emp

ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

```
SELECT DISTINCT deptno FROM emp

10
20
30
```

TOP clause :-

=> used to select top N rows from table
=> used to limit no of rows return by query

```
SYN :-    SELECT TOP <N>    *
            FROM tabname
            [WHERE cond]
```

=> display first 5 rows from emp table ?

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

=> display top 5 highest paid employee list ?

```
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
```

01-SEP-21

DML(Data Manipulation Language) commands :-

INSERT
UPDATE
DELETE
MERGE

=> all DML commands acts on table data.
=> by default in sql server every operation is auto committed (saved).
=> to stop this auto commit execute the following command

```
SET IMPLICIT_TRANSACTIONS ON
```

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

UPDATE command :-

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

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

examples :-

1 update all employees comm with 500 ?

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

2 update employee comm with 500 whose empno=7369 ?

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

3 update employee comm with 500 whose comm = null ?

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

4 update employee comm with null whose comm <> null ?

```
UPDATE emp SET comm = NULL WHERE comm IS NOT NULL
```

NULL assignment use = operator
NULL comparision use IS operator

5 increment 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%'
```

6 transfer employee from 10th dept to 20th dept ?

```
UPDATE emp SET deptno = 20 WHERE deptno=10
```

DELETE command :-

=> command used to delete row/rows from table.
=> 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 working for 30th dept and working as clerk ?

```
DELETE FROM emp WHERE deptno=30 AND job='clerk'
```

DDL(Data Definition Language) commands :-

CREATE
ALTER
DROP
TRUNCATE

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

ALTER command :-

=> command used to modify the structure

=> using ALTER we can

1 add columns
2 drop column
3 modify a column
incr/decr field size
changing datatype

Adding a column :-

=> add column gender to emp table ?

```
ALTER TABLE emp
    ADD gender CHAR(1)
```

=> after adding by default the new column is filled with NULLs , to insert data into the new column use update command.

```
UPDATE emp SET gender='m' WHERE empno=7369
```

Droping column :-

=> drop column gender from emp ?

```
ALTER TABLE emp
    DROP COLUMN gender
```

02-sep-21

Modifying a column :-
---

1 incr/decr field size
2 changing datatype

=> increase size of ename to 20 ?

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

=> decrease size of ename to 10 ?

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

ALTER TABLE emp
    ALTER COLUMN ename  VARCHAR(8)

ALTER TABLE emp
    ALTER COLUMN ename  VARCHAR(6)

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

=> change datatype of sal to money ?

ALTER TABLE emp
ALTER COLUMN sal MONEY

ALTER TABLE emp
ALTER COLUMN empno TINYINT => ERROR because values in empno column are not
with in TINYINT range.

DROP command :-

=> command used to drop table from database.
=> commnds drops table structure along with data.
=> sql server can store structure without data but sql server cannot store data without structure

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 and when memory releases data stored in the memory also deleted.

syn :- TRUNCATE TABLE <tabname>

ex :- TRUNCATE TABLE student

DROP VS DELETE VS TRUNCATE :-

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

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 condition can be used with delete	where condition 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	used by developers	used by DBAs
8	will not reset identity	will reset identity

SP_RENAME :-

=> used to change tablename or column name

syn :- SP_RENAME 'OLDNAME','NEWNAME'

=> rename table STUDENT to STUD ?

SP_RENAME 'STUDENT','STUD'

=> rename column M to MATHS in STUD table ?

SP_RENAME 'STUD.M','MATHS'

IDENTITY :-

=> used to generate sequence numbers for primary key columns
=> used to auto increment column values
=> used for numeric columns

syn :- IDENTITY(SEED,INCR)

seed => start
optional
default 1

incr => increment
optional
default 1

Example :-

```
CREATE TABLE cust
(
   cid      INT IDENTITY(100,1),
   cname    VARCHAR(10)
)

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

SELECT * FROM cust

CID	CNAME
100	A
101	B
102	C
103	D
104	E

DELETE VS TRUNCATE :-

SELECT * FROM cust

CID	CNAME
100	A
101	B
102	C
103	D
104	E

DELETE FROM cust

105	K

TRUNCATE TABLE cust

100	X

NOTE :- DELETE will not reset identity but TRUNCATE will reset identity

03-SEP-21

How to reset identity manually :-

SYN :- DBCC CHECKIDENT(tablename,reseed,value)

ex :- DBCC CHECKIDENT('cust',reseed,99)

INSERT INTO cust(cname) VALUES('PQR')

SELECT * FROM cust

CID	CNAME
100	PQR

how to insert explicit value for identity column :-

=> by default sql server will not allow explicit value for identity columns

INSERT INTO cust(cid,cname) values(200,'klm') => ERROR

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

SET IDENTITY_INSERT CUST ON

INSERT INTO cust(cid,cname) VALUES(200,'klm') => 1 row affected

SELECT * FROM cust

CID	CNAME
100	PQR
200	KLM

Built-in Functions in SQL SERVER :-

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

Types of functions :-
---

1 DATE
2 STRING
3 MATHEMATICAL
4 CONVERSION
5 SPECIAL
6 ANALYTICAL
7 AGGREGATE

DATE functions :-
---

1 GETDATE() :- returns current date,time and milliseconds

SELECT GETDATE() => 2021-09-03 16:32:42.247
---------- -------- ---

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

DATEPART(interval,DATE)

SELECT DATEPART(yy,GETDATE()) => 2021
mm => 09
dd => 03
dw => 06 (day of the week)

1 sunday
2 monday

7 saturday
dayofyear => 246 (day of the year)
hh => hour
mi => minutes
ss => seconds
qq => 3 (quarter)

1 jan-mar
2 apr-jun
3 jul-sep
4 oct-dec

w => week of the month
ww => week of the year

=> display employees joined 1980,1983,1985 ?

```
SELECT * FROM emp WHERE hiredate LIKE '1980%'
                        OR
                        hiredate LIKE '1983%'
                        OR
                        hiredate LIKE '1985%'

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

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

```
SELECT * FROM emp WHERE DATEPART(mm,hiredate) IN (01,04,12)
```

=> display employees joined on sunday ?

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

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

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

=> display employees joined in leap year ?

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

DATENAME() :- used to extract part of the date

```
        MM      DW

DATEPART  09      06

DATENAME  September   Friday
```

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

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

=> display SMITH joined on WEDNESDAY
ALLEN joined on FRIDAY ?

```
SELECT ename + ' joined on ' + DATENAME(dw,hiredate) FROM emp
```

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

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

SELECT DATEDIFF(yy,'2020-09-03',GETDATE())  =>  1
SELECT DATEDIFF(mm,'2020-09-03',GETDATE())  =>  12
```

SELECT DATEDIFF(dd,'2020-09-03',GETDATE()) => 365

=> display ENAME,EXPERIENCE in years ?

```
SELECT ename,
       DATEDIFF(yy,hiredate,GETDATE()) as experience
  FROM emp
```

06-sep-21

=> 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/years/months to/from a date.

DATEADD(interval,int,DATE)

```
SELECT DATEADD(yy,1,GETDATE())    => 2022-09-06
SELECT DATEADD(dd,10,GETDATE())   => 2021-09-16
SELECT DATEADD(mm,-2,GETDATE())   => 2021-07-06
```

scenario :-

GOLD_RATES

DATEID	RATE
2015-01-01	?
2015-01-02	?
2021-09-06	?

=> display today's gold rate ?

```
SELECT * FROM GOLD_RATES WHERE DATEID = GETDATE()
```

=> display yesterday's gold rate ?

```
SELECT * FROM GOLD_RATES WHERE DATEID = DATEADD(DD,-1,GETDATE())
```

=> display last month same day gold rate ?

```
SELECT * FROM GOLD_RATES WHERE DATEID = DATEADD(MM,-1,GETDATE())
```

=> display last year same day gold rate ?

SELECT * FROM GOLD_RATES WHERE DATEID = DATEADD(YY,-1,GETDATE())

EOMONTH() :-

=> returns last day of the month

EOMONTH(date,int)

SELECT EOMONTH(getdate(),0) => 2021-09-30

SELECT EOMONTH(getdate(),1) => 2021-10-31

SELECT EOMONTH(getdate(),-1) => 2021-08-31

Assignment :-

display current month first day ?

display next month first day ?

dislay current year first day ?

display next year first day ?

STRING functions :-

UPPER() :- converts string to uppercase

UPPER(string)

SELECT UPPER('hello') => HELLO

LOWER() :- converts string to lowercase

LOWER(string)

SELECT LOWER('HELLO') => hello

=> display empno,ename,sal ? display names in lowercase ?

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

=> in table convert names to lowercase ?

UPDATE emp SET ename = LOWER(ename)

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

LEN(string)

SELECT LEN('hello') => 5

SELECT LEN('hello welcome') => 13

=> display employees name contains 30 chars ?

SELECT * FROM emp WHERE ename LIKE '_____' ;

SELECT * FROM emp WHERE LEN(ename)=5

LEFT() :-

=> used to extract characters from left side.

LEFT(string,len)

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

SELECT LEFT('hello welcome',10) => hello welc

=> display employees name starts with 's' ?

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

RIGHT() :-

=> used to extract characters from right side

RIGHT(string,len)

SELECT RIGHT('hello welcome',4) => come

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

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

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

scenario :-

=> generate emailids for employees ?

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

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

=> store emailids in db ?

STEP 1 :- add emailid column to emp table

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

STEP 2 :- update the column with emailids

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

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

SUBSTRING(string,start,len)

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

07-sep-21

CHARINDEX() :- returns position of a character in a string.

---

syn :- CHARINDEX(char,string,[start])

=> if it finds char then returns position of if not then returns 0

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

=> display employee list name contains 'a' ?

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

SELECT * FROM emp WHERE CHARINDEX('a',ename) <> 0
```

Assignment :-

CUST

CID	CNAME
10	sachin tendulkar
11	virat kohli

=> display CID FNAME LNAME ?

hint :- using SUBSTRING,CHARINDEX

REPLICATE() :-

=> used to repeat character for given no of times

REPLICATE(char,len)

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

=> display ENAME SAL ?

800.00 ******
1600.00 *******

SELECT ename,replicate('*',LEN(sal)) as sal FROM emp

scenario :-

ACCOUNTS
ACCNO
12345678971

your a/c no XXXX8971 debited ------- ?

REPLICATE('X',4) + RIGHT(accno,4)

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

REPLACE(str1,str2,str3) => in str1 , str2 replaced with str3

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

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

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

Display ENAME SAL ?

SELECT ename,
TRANSLATE(sal,'0123456789.','*B%pT@u$#^&') as sal
FROM emp

2975.00 => %^$@&**

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

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

Mathematical Functions :-

ABS() :- returns absolute value

SELECT ABS(-10) => 10

POWER() :- returns power

SELECT POWER(3,2) => 9

SQRT() :- returns square root

SELECT SQRT(16) => 4

SQUARE() :- returns square

SELECT SQUARE(5) => 25

SIGN() :- used to check whether given number is positive or negative
---

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

FLOOR() :- rounds number to lowest

SELECT FLOOR(3.9) => 3

3----------4

CEILING() :- rounds number to highest

SELECT CEILING(3.1) => 4

3----------4

ROUND() :- rounds number based on average

ROUND(number,decimal places)

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

SELECT ROUND(38.45678,0) => 38

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

SELECT ROUND(38.55678,0) => 39

SELECT ROUND(38.45678,2) => 38.46

SELECT ROUND(38.45678,4) => 38.4568

SELECT ROUND(381,-1) => 380

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

SELECT ROUND(381,-2) => 400

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

SELECT ROUND(381,-3) => 0

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

08-sep-21

conversion functions :-

=> used to convert one datatype to another datatype.

1 CAST
2 CONVERT

1 CAST :-
---

syn :- CAST(source-expr as target-type)

SELECT CAST(10.5 AS INT) => 10

SELECT ROUND(38.4567,2) => 38.4600

SELECT CAST(ROUND(38.4567,2) AS DECIMAL(7,2)) =>38.46

=> display smith earns 800
allen earns 1600 ?

SELECT ename + ' earns ' + sal FROM emp => ERROR

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

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

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

2 CONVERT() :-

CONVERT(target-type,source-expr)

SELECT CONVERT(INT,10.5) => 10

=> diff b/w CAST & CONVERT ?

using CONVERT function we can display dates & numbers in different formats which is not possible using CAST function.

Displaying Dates in different formats :-

CONVERT(VARCHAR,DATE,STYLE-NUMBER)

Without century	With century (yyyy)	Standard	Input/Output (3)
1	101	U.S.	1 = mm/dd/yy 101 = mm/dd/yyyy
2	102	ANSI	2 = yy.mm.dd 102 = yyyy.mm.dd
3	103	British/French	3 = dd/mm/yy 103 = dd/mm/yyyy
4	104	German	4 = dd.mm.yy 104 = dd.mm.yyyy
5	105	Italian	5 = dd-mm-yy 105 = dd-mm-yyyy
6	106	-	6 = dd mon yy 106 = dd mon yyyy
7	107		7 = Mon dd, yy 107 = Mon dd, yyyy
8	108	-	hh:mi:ss
9	109		Default + milliseconds mon dd yyyy hh:mi:ss:mmmAM (or PM)
10	110	USA	10 = mm-dd-yy 110 = mm-dd-yyyy
11	111	JAPAN	11 = yy/mm/dd 111 = yyyy/mm/dd
12	112	ISO	12 = yymmdd 112 = yyyymmdd
13	113	Europe	default + milliseconds dd mon yyyy hh:mi:ss:mmm (24h)
14	114	-	hh:mi:ss:mmm (24h)

- 20 or 120 (2) ODBC canonical yyyy-mm-dd hh:mi:ss (24h)
- 21 or 25 or 121 (2) ODBC canonical (with milliseconds) default for time, date, datetime2, and datetimeoffset yyyy-mm-dd hh:mi:ss.mmm (24h)

22 - U.S. mm/dd/yy hh:mi:ss AM (or PM)

- 23 ISO8601 yyyy-mm-dd
- 126 (4) ISO8601 yyyy-mm-ddThh:mi:ss.mmm (no spaces)

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

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

Note: For a milliseconds (mmm) value of 0, the millisecond decimal value will not display. For example, the value '2012-11-07T18:26:20.000 will display as '2012-11-07T18:26:20'.
- 130 (1,2)Hijri (5) dd mon yyyy hh:mi:ss:mmmAM

In this style, mon represents a multi-token Hijri unicode representation of the full month name. This value does not render correctly on a default US installation of SSMS.
- 131 (2) Hijri (5) dd/mm/yyyy hh:mi:ss:mmmAM

=> display EMPNO ENAME HIREDATE ? display hiredates in MM/DD/YYYY format ?

```
SELECT empno,ename,CONVERT(varchar,hiredate,101) as hiredate from emp
```

=> How to input dates in different formats ?

```
INSERT INTO emp(empno,ename,hiredate)
        VALUES(9999,'ABC',CONVERT(date,'09/08/2021',101))
```

money and smallmoney styles :-

CONVERT(varchar,number,style-number)

0 No commas every three digits to the left of the decimal point, and two digits to the right of the decimal point

Example: 4235.98.

1 Commas every three digits to the left of the decimal point, and two digits to the right of the decimal point

Example: 3,510.92.

2 No commas every three digits to the left of the decimal point, and four digits to the right of the decimal point

Example: 4235.9819.

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

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

Special functions :-

ISNULL() :- used to convert null values

ISNULL(arg1,arg2)

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

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

=> display ENAME SAL COMM TOTSAL ?

TOTSAL = SAL + COMM

SELECT ename,sal,comm,sal+comm as totsal FROM emp

smith 800.00 NULL NULL
allen 1600.00 300.00 1900.00

SELECT ename,sal,comm,sal+ISNULL(comm,0) as totsal FROM emp

smith 800.00 NULL 800.00
allen 1600.00 300.00 1900.00

09-SEP-21

=> display EMPNO ENAME SAL COMM ? if comm = NULL display N/A ?

SELECT empno,ename,sal,ISNULL(CAST(comm AS VARCHAR),'N/A') as comm
FROM emp

Analytical Functions :-

RANK & DENSE_RANK :-

=> both functions are used to find ranks
=> ranking is based on one or more columns
=> for rank functions input must be sorted

syn :- RANK() OVER (ORDER BY col ASC/DESC,---)
DENSE_RANK() OVER (ORDER BY col ASC/DESC,----)

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

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

7839 king 5000.00 1
7902 ford 3000.00 2
7788 scott 3000.00 2
7566 jones 2975.00 4
7698 blake 2850.00 5

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

7839 king 5000.00 1
7902 ford 3000.00 2
7788 scott 3000.00 2
7566 jones 2975.00 3
7698 blake 2850.00 4

7782 clark 2450.00 5

=> Difference between 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 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

=> Display ranks of the employees based on sal , if salaries are same then ranking should be based on experience ?

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

7839 king 1981-11-17 5000.00 1
7902 ford 1981-12-03 3000.00 2
7788 scott 1982-12-09 3000.00 3
7566 jones 1981-04-02 2975.00 4
7698 blake 1981-05-01 2850.00 5

PARTITION BY clause :-

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

=> display ranks with in dept based on sal ?

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

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

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

7698 blake 2850.00 30 1

7499	allen	1600.00	30	2
7844	turner	1500.00	30	3
7521	ward	1250.00	30	4
7654	martin	1250.00	30	4
7900	james	950.00	30	5

11-SEP-21

ROW_NUMBER() :-

=> returns record numbers for the records return by select stmt after sorting.

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

7369	smith	800.00	1
7499	allen	1600.00	2
7521	ward	1250.00	3
7566	jones	2975.00	4
7654	martin	1250.00	5

Aggregate Functions :-

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

MAX() :- returns maximum value

MAX(arg)

SELECT MAX(sal) FROM emp => 5000

SELECT MAX(hiredate) FROM emp =>

MIN() :- returns mininum value

MIN(arg)

SELECT MIN(sal) FROM emp => 800

SUM() :- returns total

SELECT SUM(sal) FROM emp => 29025

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

SELECT CONVERT(VARCHAR,ROUND(SUM(sal),-2),1) FROM emp => 29,000.00

29000----------29050----------29025

=> calculate total sal paid to managers ?

SELECT SUM(sal) FROM emp WHERE job='MANAGER'

=> calculate total sal inlcuding commission ?

SAL COMM SAL+COMM
4000 NULL NULL
5000 500 5500
6000 NULL NULL

SUM(SAL) = 15000
SUM(SAL+COMM) = 5500

SAL COMM SAL+ISNULL(COMM,0)
4000 NULL 4000
5000 500 5500
6000 NULL 6000

SUM(SAL)= 15000
SUM(SAL+ISNULL(COMM,0)) = 15500

SELECT SUM(sal+ISNULL(comm,0)) FROM emp => 31225

AVG() :- returns average value

SELECT AVG(sal) FROM emp => 2073.2142

=> round avg(sal) to lowest ?

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

NOTE :- SUM,AVG functions cannot be applied on date,char columns can be applied only on numeric columns

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

COUNT(arg)

SELECT COUNT(empno) FROM emp => 14

SELECT COUNT(comm) FROM emp => 4 (NULLS are not counted)

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

SELECT COUNT(*) FROM emp => 14

=> diff b/w COUNT & COUNT(*) ?

COUNT function ignores nulls but COUNT(*) includes nulls

T1
F1
10
NULL
20
NULL
30

COUNT(F1) = 3

COUNT(*) = 5

=> how many employees joined in 1981 year ?

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

=> how many employees joined on sunday ?

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

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

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

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

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

=> to overcome the problem use subqueries

summary :-

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

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

math :- abs,power,sqrt,square,sign,round,ceiling,floor

conv :- cast,convert

special :- isnull

analytical :- rank,dense_rank,row_number

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

14-sep-21

CASE statement :-

=> case statement is used to implement IF-THEN-ELSE
=> case statement is similar to switch case
=> case statements are 2 types

1 simple case
2 searched case

1 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 empno ename job ?

```
      if job=CLERK display WORKER
                MANAGER          BOSS
                PRESIDENT        BIG BOSS
                others           EMPLOYEE
```

```
 SELECT empno,ename,
           CASE job
           WHEN 'CLERK'    THEN 'WORKER'
           WHEN 'MANAGER' THEN 'BOSS'
           WHEN 'PRESIDENT' THEN 'BIG BOSS'
           ELSE 'EMPLOYEE'
           END
FROM emp
```

=> increment employee salaries as follows ?

```
    if deptno=10 incr sal by 10%
                 20 incr sal by 15%
                 30                  20%
              others                 5%
```

```
UPDATE emp
SET sal = CASE deptno
              WHEN 10 then sal+(sal*0.1)
              WHEN 20 then sal+(sal*0.15)
              WHEN 30 then sal+(sal*0.2)
              ELSE sal+(sal*0.05)
              END
```

2 searched case :-

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

```
CASE
WHEN cond1 THEN return expr1
WHEN cond2 THEN return expr2
-------------------------
ELSE return expr
END
```

Example 1 :-

=> display EMPNO ENAME SAL SALRANGE ?

```
              if sal>3000    display hisal
```

sal<3000 display losal
otherwise avgsal

```
SELECT empno,ename,sal,
        CASE
        WHEN sal>3000 THEN 'Hisal'
        WHEN sal<3000 THEN 'Losal'
        ELSE 'Avgsal'
        END as salrange
FROM emp
```

Example 2 :-

STUDENT

sno	sname	s1	s2	s3
1	A	80	90	70
2	B	30	60	50

=> display SNO TOTAL AVG RESULT ?

```
SELECT sno,
        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
```

GROUP BY clause :-

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

EMP

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

--------GROUP BY---------->

10	9000
20	8000
30	4000

detailed data summarized data

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

syntax :-

```
SELECT columns
FROM tabname
[WHERE cond]
GROUP BY <col>
[HAVING cond]
```

[ORDER BY <col> ASC/DESC]

Execution :-

FROM
WHERE
GROUP BY
HAVING
SELECT
ORDER BY

=> display dept wise total salaries ?

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

FROM emp :-

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

GROUP BY deptno :-

10

1	A	5000
5	E	4000

20

2	B	3000
4	D	5000

30

3	C	4000

SELECT deptno,SUM(sal) :-

10	9000
20	8000
30	4000

=> display job wise no of employees ?

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

=> display year wise no of employees joined ?

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

=> display day of the week wise no of employees joined ?

```
SELECT DATENAME(dw,hiredate) as day,COUNT(*) as cnt
FROM emp
GROUP BY DATENAME(dw,hiredate)
```

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

```
SELECT DATEPART(qq,hiredate) as qrt,COUNT(*) as cnt
FROM emp
WHERE DATEPART(yy,hiredate)=1981
GROUP BY DATEPART(qq,hiredate)
```

15-sep-21

=> display the dept where more than 3 employees working ?

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

=> above query returns error because SQL SERVER cannot calculate dept wise count before group by
and it can calculate only after group by , so apply the condition count(*) > 3 after group by using HAVING condition

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

WHERE VS HAVING :-

	WHERE	HAVING
1	used to select specific rows	used to select specific groups
2	conditions applied before group by	conditions applied after group by
3	use where clause if cond doesn't contain aggregate function	use having clause if condition contains aggregate function

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

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

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        1100
     MANAGER         2975

30   CLERK        950
     MANAGER         2850
     SALESMAN     5600
```

scenario :-

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

1 display state wise no persons where state = AP,TS,KA,KL,TN and no of persons > 5crores ?

```
SELECT STATE,COUNT(*)
FROM PERSONS
WHERE STATE IN ('AP','TS','KA','KL','TN')
GROUP BY STATE
HAVING COUNT(*) > 50000000
```

2 display state wise and with in state gender wise no of persons ?

```
SELECT STATE,GENDER,COUNT(*)
FROM PERSONS
GROUP BY STATE,GENDER
ORDER BY STATE ASC
```

```
AP   MALE    ?
     FEMALE ?

AR    MALE   ?
      FEMALE ?

AS    MALE
      FEMALE ?

TS   MALE   ?
     FEMALE ?
```

ROLLUP & CUBE :-

=> both functions are used to calculate subtotals and grand totals

GROUP BY ROLLUP(COL1,COL2,----)
GROUP BY CUBE(COL1,COL2,------)

ROLLUP :-

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

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

```
NULL    NULL        29225.00   => grand total
10  CLERK       1300.00
10  MANAGER         2450.00
10  PRESIDENT   5000.00
10  NULL        8750.00 => subtotal
20  ANALYST         7000.00
20  CLERK       1100.00
20  MANAGER         2975.00
20  NULL        11075.00 => subtotal
30  CLERK       950.00
30  MANAGER         2850.00
30  SALESMAN    5600.00
30  NULL        9400.00 => subtotal
```

CUBE :-

=> CUBE display 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  ANALYST         7000.00   => job subtotal
NULL    CLERK       3350.00   => job subtotal
NULL    MANAGER         8275.00
NULL    PRESIDENT   5000.00
NULL    SALESMAN    5600.00
NULL    NULL        29225.00 => grand total
10  NULL        8750.00   => dept subtotal
10  PRESIDENT   5000.00
10  MANAGER         2450.00
10  CLERK       1300.00
20  CLERK       1100.00
20  MANAGER         2975.00
20  NULL        11075.00 => dept subtotal
20  ANALYST         7000.00
```

30 NULL 9400.00 => dept subtotal
30 SALESMAN 5600.00
30 MANAGER 2850.00
30 CLERK 950.00

Assignement :-

1

PERSONS
AADHARNO NAME GENDER AGE ADDR CITY STATE

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

2
SALES
DATEID PRODID CUSTID QTY AMOUNT
2015-01-01 100 10 1 5000

2021-09-16 110 20 1 2000

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

summary :-

importance of group by
writing group by queries
where vs having
grouping based on multiple columns
rollup & cube

16-sep-21 INTEGRITY CONSTRAINTS

=> Integrity Constraints are rules to maintain data integrity i.e. data quality
=> Integrity Constraints are used to prevent users from entering invalid data.
=> Integrity Constraints are used to enforce rules like min bal must be 1000

Integrity Constraints :-
---

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

=> above constraints can be declared in two ways

1 column level
2 table level

column level :-

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

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

NOT NULL :-

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

EX :-

```
create table emp11
 (
    empno int,
    ename varchar(10) not null
 )

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

UNIQUE :-

=> UNIQUE constraint doesn't accept duplicates

ex :-

```
CREATE TABLE emp12
(
    empno int,
    emailid    varchar(20) UNIQUE
 )

INSERT INTO emp12 VALUES(100,'abc@gmail.com')
INSERT INTO emp12 VALUES(101,'abc@gmail.com')    => ERROR
INSERT INTO emp12 VALUES(102,NULL)                => ACCEPTED
INSERT INTO emp12 VALUES(103,NULL)                => ERROR
```

PRIMARY KEY :-

=> PRIMARY KEY doesn't accept duplicates and nulls.

=> PRIMARY KEY is the combination of UNIQUE & NOT NULL

PRIMARY KEY = UNIQUE + NOT NULL

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

Ex :-

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

INSERT INTO emp13 VALUES(100,'abc')
INSERT INTO emp13 VALUES(100,'pqr') => error
INSERT INTO emp13 VALUES(null,'pqr') => error
```

=> using primary key column we can uniquely identify the records.

=> only one primary key is allowed per table , if we want two primary keys then declare one column with primary key and antother column with UNIQUE & NOT NULL.

```
CREATE TABLE cust
 (
    custid       int    PRIMARY KEY,
    name         varchar(10) NOT NULL,
    aadharno     bigint UNIQUE NOT NULL,
    panno         char(10) UNIQUE NOT NULL
 )
```

candidate key :-

=> a field which is eligible for primary key is called candidate key.

ex :- VEHICLES
VEHNO NAME MODEL COST CHASSISNO

candidate keys :- VEHNO,CHASSISNO
primary key :- VEHNO
secondary key :- CHASSISNO
or
alternate key

=> while creating table secondary keys are declared with UNIQUE & NOT NULL.

17-sep-21

CHECK constraint :-

=> use check constraint when rule based on condition.

syn :- CHECK(condition)

Example 1 :- sal must be min 3000

```
CREATE TABLE emp14
(
   empno int,
   ename varchar(10),
   sal     money CHECK(sal>=3000)
)
```

```
INSERT INTO emp14 VALUES(100,'A',5000)
INSERT INTO emp14 VALUES(101,'B',1000) => ERROR
INSERT INTO emp14 VALUES(102,'C',NULL) => ACCEPTED
```

NOTE :- check constraint allows nulls

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

```
        gender char(1) check(gender in ('m','f'))
```

Example 3 :- pwd must be min 8 chars

```
        pwd     varchar2(12)    check(len(pwd)>=8)
```

Example 4 :- emailid must contain '@'
emailid 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	NAME	DURATION	COST	CLIENT
100	A	5 YEARS	150	TATA MOTORS
101	B	3 YEARS	80	DBS
102	C	4 YEARS	120	L&T

EMP

EMPID	ENAME	SAL	PROJID	REFERENCES PROJECTS(PROJID)
1	A	5000	100	
2	B	4000	101	
3	C	7000	999	=> NOT ACCEPTED
4	D	3000	100	

5 E 2000 NULL

=> in the above example relationship between projects and emp is one to many (1:M) , so add foreign key to many side table i.e. emp.

=> values entered in foreign key column shoule match with values entered in primary key column

=> fk allows duplicates and nulls

=> after declaring foreign key a relationship is established between two tables called parent/child relationship

=> primary key table is parent(projects) and foreign key table is child (emp) .

```
CREATE TABLE projects
(
   projid    int PRIMARY KEY,
   name      varchar(10),
   duration varchar(15),
   cost      money,
   client    varchar(20)
)

INSERT INTO projects VALUES(100,'A','5 YEARS',150,'TATA MOTORS')
INSERT INTO projects VALUES(101,'B','3 YEARS',80,'DBS')

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

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

18-sep-21

how to establish one to one (1:1) relationship :-

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

Example :-

DEPT
DEPTNO DNAME
10 HR
20 IT

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

=> in the above example one dept is managed by one manager and one manager manages one dept so the
relationship between dept & mgr is one to one

```
CREATE TABLE dept55
 (
  deptno int primary key,
  dname varchar(10)
 )

 INSERT INTO dept55 VALUES(10,'HR'),(20,'IT')

CREATE TABLE manager
(
  mgrno   int primary key,
  mname   varchar(10),
  deptno int references dept(deptno) unqiue
 )

 INSERT INTO manager VALUES(1,'A',10)
 INSERT INTO manager VALUES(2,'B',10) => ERROR
```

DEFAULT constraint :-

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

ex :- hiredate date default getdate()

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

```
  CREATE TABLE emp66
  (
   empno int ,
   hiredate date default getdate())
  )


   INSERT INTO emp66(empno) VALUES(100)
   INSERT INTO emp66 VALUES(101,'2021-01-01')
   INSERT INTO emp66 VALUES(102,NULL)

   SELECT * FROM emp66

   empno    hiredate
   100      2021-09-18
   101      2021-01-01
   102      null
```

table level :-

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

```
CREATE TABLE <tabname>
(
   col1 datatype(size),
   col2 datatype(size),
   ------------------,
        constraint(col1,col2,--)
);
```

Declaring check constraint at table level :-

```
PRODUCTS
PRODID        NAME    MFD_DT         EXP_DT
100           A       2021-09-18 2021-01-01    INVALID
```

RULE :- EXP_DT > MFD_DT

```
CREATE TABLE PRODUCTS
(
   prodid int,
   pname   varchar(10),
   mfd_dt date ,
   exp_dt date ,
        check(exp_dt>mfd_dt)
)
```

20-sep-21

composite primary key :-

=> some tables we can't uniquely identify the records by using single column and we need combination
of columns to uniquely identity and that combination should be declared with primary key.

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

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

=> composite primary key declared at table level.

Example :-

```
ORDERS                                          PRODUCTS
ordid    ord_dt    del_dt    cid          prodid    pname   price
1000     15/09     20/09     10             100     A     5000
1001     18/09     22/09     11             101     B     8000
1002     19/09     25/09     12             102     C     9000


ORDER_DETAILS
ordid    prodid    qty
1000     100   1
1000     101   1
```

```
1000      102   1
1001      100   1
1001      101   1

CREATE TABLE orders
(
   ordid int primary key,
   ord_dt date,
   del-dt date,
   cid int
)

 INSERT INTO orders VALUES(1000,'2021-09-15',GETDATE(),10)
 INSERT INTO orders VALUES(1001,'2021-09-18','2021-09-22',10)

 CREATE TABLE products
 (
    prodid int primary key,
    pname    varchar(10),
    price    money
 )

 INSERT INTO products VALUES(100,'A',5000)
 INSERT INTO products VALUES(101,'B',3000)

 CREATE TABLE order_details
 (
  ordid    int references orders(ordid),
  prodid int references products(prodid) ,
  qty int,
       primary key(ordid,prodid)
 )

  INSERT INTO order_details VALUES(1000,100,1)
  INSERT INTO order_details VALUES(1000,101,1)
  INSERT INTO order_details VALUES(1000,100,1) => ERROR

 Question :-

 which of the following constraints cannot be declared at table level ?

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

 ANS :- NOT NULL

 Assignment 1 :-
 ----------

 ACCOUNTS
 ACCNO NAME    ACTYPE BAL

 Rules :-
```

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

TRANSACTIONS
TRID TTYPE TDATE TAMT ACCNO

Rules :-

1 trid must be automatically generated
2 ttype must be 'w'or 'd'
3 tdate must be default current date
4 tamt must be multiple of 100
5 accno should match with accounts table accno
6 accno should not be null

Assignment 2 :-

SALES
DATEID PRODID CUSTID QTY AMOUNT
20- 100 10 1 3000
20- 100 11 1 3000
20- 101 10 1 2000
21- 100 10 1 3000

=> identity primary key and write create table script ?

Assignment 3 :-

STUDENT COURSE
SID SNAME CID CNAME
1 A 10 SQL
2 B 11 .NET

=> establish many to many relationship between two tables ?

21-sep-21

Adding constraints to existing table :-

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

```
CREATE TABLE emp66
(
 empno int,
 ename varchar(10),
 sal money,
 dno int
)
```

Adding primary key :-

=> primary key cannot be added to nullable columns , to add primary key first change

column to NOT NULL
then add primary key.

=> add primary key to column empno ?

step 1 :- ALTER TABLE emp66
ALTER COLUMN empno INT NOT NULL

step 2 :- ALTER TABLE emp66
ADD PRIMARY KEY(empno)

Adding check constraint :-

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

ALTER TABLE emp66
ADD CHECK(sal>=3000)

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

=> above command returns error becauase in emp table some of the employee salaries are less than 3000 ,
while adding constraint sql server also validates existing data.

WITH NOCHECK :-

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

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

Adding foreign key :-

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

ALTER TABLE emp66
ADD FOREIGN KEY(dno) REFERENCES dept(deptno)

changing from NULL to NOT NULL :-

ALTER TABLE emp66
ALTER COLUMN ename VARCHAR(10) NOT NULL

Droping constraints :-

ALTER TABLE <TABNAME>
DROP CONSTRAINT <NAME>

=> drop check constraint in emp66 table ?

```
ALTER TABLE emp66
    DROP CONSTRAINT CK__emp66__sal__300424B4
```

=> drop primary key in dept table ?

```
ALTER TABLE DEPT
        DROP CONSTRAINT PK__DEPT__E0EB08D7753DBA0D    => ERROR

DROP TABLE DEPT    => ERROR

TRUNCATE TABLE DEPT => ERROR
```

NOTE :- pk constraint cannot be dropped if referend by some fk
pk table cannot be dropped if referenced by some fk
pk table cannot be truncated if referenced by some fk

DELETE rules :-

ON DELETE NO ACTION (DEFAULT)
ON DELETE CASCADE
ON DELETE SET NULL
ON DELETE SET DEFAULT

=> above delete rules are declared with foreign key
=> delete rules specifies how child rows are affected if parent row is deleted

ON DELETE NO ACTION :-

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

```
CREATE TABLE dept77
 (
 dno int primary key,
 dname varchar(10)
 )

 INSERT INTO dept77 VALUES(10,'HR'),(20,'IT')

CREATE TABLE emp77
(
   empno int primary key,
   ename varchar(10),
   dno int references dept77(dno)
)

INSERT INTO emp77 VALUES(1,'A',10),(2,'B',10)

SELECT * FROM DEPT77

DNO     DNAME
10 HR
20 IT
```

```
SELECT * FROM EMP77

ENO     ENAME   DNO
1    A     10
2    B     10

DELETE FROM dept77 WHERE dno=10      => ERROR

DELETE FROM dept77 WHERE dno=20      => 1 row affected

ON DELETE CASCADE :-
--------------------

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

CREATE TABLE dept77
  (
  dno int primary key,
  dname varchar(10)
  )

  INSERT INTO dept77 VALUES(10,'HR'),(20,'IT)

CREATE TABLE emp77
(
    empno int primary key,
    ename varchar(10),
    dno int references dept77(dno)
                ON DELETE CASCADE
)

INSERT INTO emp77 VALUES(1,'A',10),(2,'B',10)

SELECT * FROM DEPT77

DNO     DNAME
10  HR
20  IT

SELECT * FROM EMP77

ENO     ENAME   DNO
1    A     10
2    B     10

DELETE FROM DEPT77 WHERE DNO=10    => 1 ROW AFFECTED


SELECT * FROM EMP77

ENO     ENAME   DNO

ON DELETE SET NULL :-
----------------------

  => if parent row is deleted then child rows are not deleted but fk will be set to null
```

```
CREATE TABLE dept77
   (
   dno int primary key,
   dname varchar(10)
   )

   INSERT INTO dept77 VALUES(10,'HR'),(20,'IT')

  CREATE TABLE emp77
  (
     empno int primary key,
     ename varchar(10),
     dno int references dept77(dno)
                 ON DELETE SET NULL
  )

  INSERT INTO emp77 VALUES(1,'A',10),(2,'B',10)

  SELECT * FROM DEPT77

  DNO     DNAME
  10  HR
  20  IT

  SELECT * FROM EMP77

  ENO     ENAME   DNO
  1    A     10
  2    B     10

  DELETE FROM dept77 WHERE dno=10    =>    1 row affected

  SELECT * FROM EMP77

  ENO     ENAME   DNO
  1    A     NULL
  2    B     NULL

ON DELETE SET DEFAULT :-
-----------------------

=> if parent row is deleted then it is deleted but child rows are not deleted but fk will be set to
      default value.

  CREATE TABLE dept77
   (
   dno int primary key,
   dname varchar(10)
   )

   INSERT INTO dept77 VALUES(10,'HR'),(20,'IT')

  CREATE TABLE emp77
  (
     empno int primary key,
     ename varchar(10),
     dno int DEFAULT 20
```

```
                references dept77(dno)
                ON DELETE SET DEFAULT
 )

 INSERT INTO emp77 VALUES(1,'A',10),(2,'B',10)

SELECT * FROM DEPT77

 DNO     DNAME
 10 HR
 20 IT

 SELECT * FROM EMP77

 ENO     ENAME   DNO
 1   A     10
 2   B     10

   DELETE FROM DEPT77 WHERE DNO=10 => 1 ROW AFFECTED

 SELECT * FROM EMP77

 ENO     ENAME   DNO
 1   A     20
 2   B     20


 ACCOUNTS
 ACCNO NAME BAL
 100

 LOANS
 ID       TYPE    AMOUNT ACCNO
 1   H     30    100
 2   C     10    100

TRANSACTIONS
TRID      TTYPE    TDATE    TAMT     ACCNO
1     W          200 100
2     D          5000       100


PROJECTS
projid     pname

101 b

emp
empno   ename    projid
1     a     null
2     b     null

UPDATE rules :-
----------------

ON UPDATE NO ACTION (DEFAULT)
ON UPDATE CASCADE
```

ON UPDATE SET NULL
ON UPDATE SET DEFAULT

summary :-

importance of constraints
types of constraints
declaring constraints
 column level
 table level
adding constraints to existing table
droping constraints
deletes rules
update rules

22-sep-21 JOINS

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

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

Example :-

ORDERS CUSTOMERS

ordid	ord_dt	del_dt	custid	cid	cname	caddr
1000	10-	20-	10	10	a	hyd
1001	11-	21-	11	11	b	hyd

report :-

ordid	ord_dt	del_dt	cname	caddr
1000	10-	20-	A	HYD
1001	11-	21-	B	HYD

Types of joins :-
---

1 Equi Join or Inner join
2 Outer Join
 left join
 right join
 full join
3 Non equi join
4 self join
5 cross join or cartesian join

Equi Join / Inner join :-

=> To perfomr equi 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.

=> equi join is performed on the common column with same datatype.

```
SELECT columns
FROM tabnames
WHERE join condition
```

join condition :-

=> based on the given join condition sql server joins records of two tables
=> join condition determines which record of 1st table joined with which record of 2nd table

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

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

Example :-

EMP

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

DEPT

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

=> display EMPNO ENAME SAL DNAME LOC ?
---------- ----------
EMP DEPT

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

1	A	5000	ACCOUNTS	???
2	B	4000	RESEARCH	???
3	C	3000	SALES	???
4	D	4000	ACCOUNTS	???

=> display EMPNO ENAME SAL DEPTNO DNAME LOC ?

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

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

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

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

joining more than 2 tables :-

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

```
SELECT columns
FROM tab1,
       tab2,
       tab3,
       ---
WHERE join cond1
        AND
        join cond2
        AND
        join cond3
```

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_NAME ?

-----	------	----------	----------
emp	dept	locations	countries

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

```
AND
d.locid = l.locid
AND
l.country_id = c.country_id
```

23-sep-21

join styles :-
---

1 Native style (sql server)
2 ANSI style

ANSI style :-

=> Adv of ANSI style is portability
=> Native style doesn't gurantees portability but ANSI style gurantees portability
=> join queries can be easily migrated from one db to another db
=> in ANSI style tablenames are seperated by keywords and use ON clause for join conditions
instead of where clause.

Display ENAME DNAME ?

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

=> display ENAME DNAME working at NEW YORK loc ?

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

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

OUTER JOIN :-

=> equi join returns only matching records but will not return unmatched records , to get unmatched records also perform outer join.

Example:-

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

=> outer join is possible in ANSI style

=> outer join is 3 types

1 left join
2 right join
3 full join

left join :-

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

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

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

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

right join :-

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

```
      A     ACCOUNTS
B     RESEARCH
C     SALES
D     ACCOUNTS
   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     ACCOUNTS
B     RESEARCH
C     SALES
D     ACCOUNTS
```

E NULL => unmatched from emp
NULL OPERATIONS => unmatched from dept

=> Displaying unmatched records from emp ?

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

E NULL

=> display unmatched records from dept ?

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

=> display unmatched records from both tables ?

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

Assignment :-

PROJECTS

projid	name	duration
100		
101		
102		

EMP

empid	ename	sal	projid
1			100
2			101
3			null

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

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

=> display employees not working for any project ?

=> display projects where not employee assigned to it ?

Non Equi Join :-

=> Non equi join is performed when tables are not sharing a common field.

=> this join is called non equi join because here join condition is based on > < between operators

Example :-

EMP

EMPNO	ENAME	SAL
1	A	5000
2	B	2500
3	C	1000
4	D	3000
5	E	1500

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	2000	3
C	1000	1
D	3000	3
E	1500	2

=> 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 ;
```

24-SEP-21

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

Ex :-

EMP

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

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

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

=> 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

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

EMP Y

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

=> display ENAME MGRNAME ?

```
SELECT x.ename,y.ename as manager
  FROM emp x join emp y
    ON x.mgr = y.empno
```

SMITH	FORD
ALLEN	BLAKE
WARD	BLAKE

=> display list of 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
```

Assignment :-

1 display employees joined before their manager
2 display no of employees working under each manager ?
3 display manager names where no of employees reporting is > 3 ?
4 display employees earning same salary ?
5 display employee joined same date ?
6

TEAMS

ID	COUNTRY
1	IND
2	AUS
3	RSA

=> Write a query to display following output ?

IND VS AUS
IND VS RSA
AUS VS RSA

7 display ENAME DNAME GRADE MNAME ?

-----	-----	----	------
EMP	DEPT	SALGRADE	EMP

CROSS JOIN OR 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 each record of table1 joined with each and every
record table2.

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

GROUP BY & JOIN :-

=> display dept wise no of employees , in output display dept names ?

```
SELECT d.dname,COUNT(e.empno) as cnt
FROM emp e INNER JOIN dept d
    ON e.deptno = d.deptno
 GROUP BY d.dname
```

=> display no of employees reporting to each manager ?

```
SELECT y.ename as manager,COUNT(x.ename) as cnt
  FROM emp x JOIN emp y
    ON x.mgr = y.empno
GROUP BY y.ename
```

27-sep-21

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
=> result of inner query is input to outer query
=> use subquery when where condition is based on unknown value

Types of subqueries :-
---

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

1 single row subqueries :-

=> if subquery 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 :-

1 display employees earning more than blake ?

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

2 display employees who are senior to king ?

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

3 display employee name earning max salary ?

```
SELECT ename
FROM emp
WHERE sal = MAX(sal)    => ERROR    (aggregate functions are not allowed in where clause)

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

4 display employee name having max experience ?

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

5 delete the employee having max experience ?

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

6 swap employee salaries whose empno = 7499,7521 ?

```
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)
```

multirow subqueries :-

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

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

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

Examples :-

=> display employee list whose job is same as   smith,blake ?

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

ANY operator :-

=> operator used to compare with any of the value

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

    IF X=800       FALSE
       1500        TRUE
```

4500 TRUE

WHERE X < ANY(1000,2000,3000)

IF X=800 TRUE
1500 TRUE
4500 FALSE

ALL operator :-

=> use ALL operator when comparision with all 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

=> display employees earning more than all managers ?

```
select *
from emp
where sal > ALL(select sal from emp where job='manager')
```

SINGLE	MULTI		
=	IN		
>		>ANY	>ALL
<		<ANY	<ALL

28-sep-21

co-related subqueries :-

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

=> in co-related subquery execution starts from outer query and inner query is executed for each row return by outer query.

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

1 returns row from outer query
2 pass value to inner query
3 executes inner query
4 inner query output is passed to outer query

5 executes outer query where condition

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

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

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

1	A	5000	10	5000 > (SELECT AVG(sal) FROM emp WHERE deptno = 10)	4000 TRUE
2	B	3000	20	3000 > (SELECT AVG(sal) FROM emp WHERE deptno = 20)	4500 FALSE
3	C	4000	30	4000 > (SELECT AVG(sal) FROM emp WHERE deptno = 30)	4000 FALSE
4	D	6000	20	6000 > (SELECT AVG(sal) FROM emp WHERE deptno = 20)	4500 TRUE
5	E	3000	10	3000 > (SELECT AVG(sal) FROM emp WHERE deptno = 10)	4000 FALSE

=> display employees earning max(sal) in their dept ?

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

1	A	5000	10	5000 = (SELECT MAX(sal) FROM emp WHERE deptno = 10)	5000 TRUE
2	B	3000	20	3000 = (SELECT MAX(sal) FROM emp WHERE deptno = 20)	6000 FALSE
3	C	4000	30	4000 = (SELECT MAX(sal) FROM emp WHERE deptno = 30)	4000 TRUE

=> display top 3 max salaries ?

```
SELECT DISTINCT A.SAL
FROM EMP A
WHERE 3 > (SELECT COUNT(DISTINCT B.SAL)
           FROM EMP B
           WHERE A.SAL<B.SAL)
ORDER BY SAL DESC
```

EMP A SAL	EMP B SAL		
5000	5000	3 > (0)	TRUE
1000	1000	3 > (4)	FALSE
3000	3000	3 > (2)	TRUE

2000	2000	3 > (3)	FALSE
4000	4000	3 > (1)	TRUE

=> display 3rd max salary ?

```
SELECT DISTINCT A.SAL
FROM EMP A
WHERE (3-1) =   (SELECT COUNT(DISTINCT B.SAL)
                 FROM EMP B
                 WHERE A.SAL<B.SAL)
ORDER BY SAL DESC
```

Derived tables :-

=> subqueries in FROM clause are called derived tables

```
syn :-  SELECT columns
          FROM (SELECT STATEMENT)   <ALIAS>
          WHERE CONDITION
```

=> subquery output acts like a table for outer query

=> derived tables are used in following scenarios

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

Example 1 :-

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

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

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

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

above query returns error because in where clause we can't refer column aliases because where clause is executed before select , to overcome this problem use derived tables.

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

=> display employee details earning 5th max salary ?

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

Assignment :- display top 3 employees in each dept ?

Example 2 :- display first 5 rows from emp ?

```
SELECT *
FROM ( SELECT empno,ename,sal,
          ROW_NUMBER() OVER (ORDER BY empno ASC) as rno
     FROM emp ) E
WHERE rno<=5

WHERE rno=5

WHERE rno IN (5,7,10)

WHERE rno BETWEEN 5 AND 10

WHERE rno%2=0
```

Example 3 :- delete first 5 rows from emp table ?

```
DELETE
FROM ( SELECT empno,ename,sal,
          ROW_NUMBER() OVER (ORDER BY empno ASC) as rno
     FROM emp ) E
WHERE rno<=5      => ERROR
```

NOTE :- in dervied tables outer query cannot be DML(INSERT/UPDATE/DELETE) and it must be
SELECT statement , to overcome this problem use CTEs

CTE :-

=> CTEs stands for common table expression
=> CTE is a named result set and we can use this name in another queries like SELECT/INSERT/UPDATE/DELETE
=> using CTEs we can simplify the complex operations
=> CTEs are also like derived tables but in derived tables outer query cannot be DML command but in
CTEs outer query can be DML command.
syntax :-

```
WITH <NAME>
 AS
   (SELECT STATEMENT),
 <NAME>
  AS
    (SELECT STATEMENT---)
```

SELECT/INSERT/UPDATE/DELETE

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

```
WITH E
AS
  (SELECT empno,ename,sal,
     ROW_NUMBER() OVER (ORDER BY sal DESC) as rno
   FROM emp)
DELETE FROM E WHERE RNO<=5
```

Example 2 :- delete duplicate records from table ?

```
EMP22
ENO   ENAME  SAL
1  A    5000
2  B    6000
1  A    5000 => duplicate row
2  B    6000 => duplicate row
3  C    7000
```

step 1 :-
```
SELECT   eno,ename,sal,
         ROW_NUMBER() OVER (PARTITION BY eno,ename,sal ORDER BY eno ASC) as rno
  FROM emp22
```

```
1   A    5000   1
1   A    5000   2

2   B    6000   1
2   B    6000   2

3   C    7000   1
```

step 2 :- 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 emp22)
DELETE FROM E WHERE RNO> 1
```

Question 1 :-

```
T1      T2
F1      C1
1       A
2       B
3       C
```

=> join the two tables and display following output ?

```
1    A
2    B
```

3 C

Question 2 :-

T1
AMT
1000
-500
2000
-3000
4000
-1500

output :-

pos	neg
1000	-500
2000	-3000
4000	-1500

Question 3 :-

CLERK	MANAGER	SALESMAN
SMITH	BLAKE	ALLEN
JONES	CLARK	WARD
	JONES	MARTIN

30-sep-21

Database Transactions :-

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

ex :- money transfer

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

update1 (bal=bal-1000)	update2 (bal=bal+1000)	
successful	failed	invalid
failed	successful	invalid
successful	successful	valid
failed	failed	valid

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

=> the following commands provided by sql server to handle transactions called TCL

commands

1 commit => to save transaction
2 rollback => to cancel transaction
3 save transaction => to cancel transaction upto to some point

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

=> in sql server a txn begins implicitly with dml/ddl commands and ends implicitly with commit

=> user can also start txn explicitly with "BEGIN TRANSACTION" command and ends explicitly with COMMIT/ROLLBACK

Example 1 :-

```
CREATE TABLE a(a int)
BEGIN TRANSACTION                => TXN STARTS T1
INSERT INTO a VALUES(10)
INSERT INTO a VALUES(20)
INSERT INTO a VALUES(30)
INSERT INTO a VALUES(40)
INSERT INTO a VALUES(50)
COMMIT                           => TXN ENDS
```

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

Example 2 :-

```
CREATE TABLE a(a int)
BEGIN TRANSACTION                => TXN STARTS T1
INSERT INTO a VALUES(10)
INSERT INTO a VALUES(20)
INSERT INTO a VALUES(30)
INSERT INTO a VALUES(40)
INSERT INTO a VALUES(50)
ROLLBACK                         => TXN ENDS
```

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

Example 3 :-

```
CREATE TABLE a(a int)
BEGIN TRANSACTION                => txn starts T1
INSERT INTO a VALUES(10)
INSERT INTO a VALUES(20)
INSERT INTO a VALUES(30)
COMMIT                           => txn ends
INSERT INTO a VALUES(40)
INSERT INTO a VALUES(50)
ROLLBACK
```

output :-

```
select * from a
```

10

20
30
40
50

SAVE TRANSACTION :-

=> we can declare save transaction and we can rollback upto the save transaction.
=> using save transaction we can cancel part of the transaction

Example 1 :-

```
CREATE TABLE a(a int)
BEGIN TRANSACTION
INSERT INTO a VALUES(10)
INSERT INTO a VALUES(20)
SAVE TRANSACTION ST1
INSERT INTO a VALUES(30)
INSERT INTO a VALUES(40)
SAVE TRANSACTION ST2
INSERT INTO a VALUES(50)
INSERT INTO a VALUES(60)
ROLLBACK    TRANSACTION ST2

  SELECT * FROM A
```

10
20
30
40

Example 2 :-

```
CREATE TABLE a(a int)
BEGIN TRANSACTION
INSERT INTO a VALUES(10)
INSERT INTO a VALUES(20)
SAVE TRANSACTION ST1
INSERT INTO a VALUES(30)
INSERT INTO a VALUES(40)
SAVE TRANSACTION ST2
INSERT INTO a VALUES(50)
INSERT INTO a VALUES(60)
ROLLBACK    TRANSACTION ST1

SELECT * FROM A
```

10
20

Database Security :-
---

1 logins => provides security at server level
2 users => provides security at db level
3 privileges => provides security at table level

4 views => provides security at row & col level

SERVER (LOGIN)
DATABASE (USERS)
TABLES (PRIVILEGES)
ROWS & COLS (VIEWS)

DB objects / SCHEMA objects :-
---

1 TABLES
2 VIEWS
3 SEQUENCES
4 INDEXES

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

=> a view is a representation query

=> views are created

1 to provide security
2 to reduce complexity

=> views provides another level of security called row & col level , with the help of views we can grant specific rows and columns to users instead of granting whole table.

=> views are 2 types

1 simple views
2 complex views

simple views :-

=> if view created on single table then it is called simple view.

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

Example :-

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

=> when the above command submitted to sql server , it creates view v1 and stores query but not query output

SELECT * FROM V1

=> when the above query submitted then sql server rewrite the query as follows

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

Granting permissions on view to user :-

DBO :-

GRANT SELECT,INSERT,UPDATE,DELETE ON V1 TO KUMAR

KUMAR :-

1 SELECT * FROM V1

2 INSERT INTO V1 VALUES (8888,'suresh','clerk',getdate(),20)

3 UPDATE V1 SET JOB='MANAGER' WHERE EMPNO=8888

4 UPDATE V1 SET SAL=5000 WHERE EMPNO=8888 => ERROR

ROW LEVEL SECURITY :-

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

GRANT SELECT,INSET,UPDATE,DELETE ON V2 TO KUMAR

Complex views :-

=> a view said to be complex view

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

=> with the help of views complex queries are converted into simple queries

Example 1 :-

```
CREATE VIEW CV1
AS
SELECT e.empno,e.ename,e.sal,d.deptno,d.dname,d.loc
FROM emp e INNER JOIN dept d
   ON e.deptno = d.deptno
```

=> after creating view whenever we want data from emp & dept tables instead of writing

join query
write the simple query as follows

```
SELECT * FROM CV1
```

Example 2 :-

```
CREATE VIEW CV2
AS
SELECT d.dname,MIN(e.sal) as minsal,
                    MAX(e.sal) as maxsal,
                    SUM(e.sal) as totsal,
                    COUNT(e.empno) as cnt
FROM emp e INNER JOIN dept d
   ON e.deptno = d.deptno
GROUP BY d.dname
```

=> whenever we want dept wise summary then simply execute the following query

```
SELECT * FROM CV2
```

=> difference between simple and complex views ?

	simple	complex
1	based on single table	based on multiple tables
2	query performs simple operations	query performs complex operations like joins,group by etc
3	always updatable i.e. allows dmls	doesn't allow dmls

=> display list of views created by user ?

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS
```

Droping views :-

```
DROP VIEW V1
```

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

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

SCHEMABINDING :-

=> if view created with schemabinding then sql server will not allow user to drop tables if any view
exists on the table , to drop table first we need to drop view.

rules :-

1 "*" is not allowed in schemabinding

2 tablename should be prefixed with schemaname

```
CREATE VIEW V10
WITH SCHEMABINDING
AS
SELECT deptno,dname,loc FROM dbo.DEPT


DROP TABLE DEPT    => ERROR
```

04-oct-21

SEQUENCES :-

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

syn :-

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

Example :-

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

=> use above sequence to generate values for sid in student table ?

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

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

SELECT * FROM STUDENT

SID   SNAME
1     A
2     B
3     C
```

4 D
5 E

calling sequence in update command :-

CREATE SEQUENCE S2
START WITH 100
INCREMENT BY 1
MAXVALUE 1000

=> use above sequence to update employee numbers ?

UPDATE emp SET empno = NEXT VALUE FOR S2

calling sequence in expressions :-

INVOICE
INVNO INV_DT
KLM/1021/1 ??
KLM/1021/2 ??

```
CREATE SEQUENCE S3
START WITH 1
INCREMENT BY 1
MAXVALUE 1000

CREATE TABLE INVOICE
(
   INVNO    VARCHAR(20),
   INV_DT DATETIME
)

INSERT INTO INVOICE
        VALUES('KLM/' + FORMAT(getdate(),'MMyy') + '/' + CAST(next value for s3 AS
VARCHAR),GETDATE())
```

how to reset sequence :-
---

1 using alter command
2 using cycle option

using alter command :-

ALTER SEQUENCE S2 RESTART WITH 100

using cycle option :-

=> by default sequence is created with NOCYCLE i.e. it starts from start with and generates upto max
and after reaching max then it stops.

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

```
CREATE SEQUENCE S5
START WITH 1
INCREMENT BY 1
MAXVALUE 5
MINVALUE 1
CYCLE
```

=> above sequence starts from 1 and everytime value is incremented by 1 and generates upto 5 and
after reaching 5 then it will be reset to 1

INDEXES :-

=> index is also a db object created to improve performance of data accessing

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

=> sql server follows below steps to execute any query submitted to sql server

1 parsing
2 optimization
3 execution

parsing :-
---

1 checks syntax
2 checks semantics
checks whether table exists or not
checks whether columns belongs to table or not
checks whether user has got permission to access table or not

optimization :-

=> prepares plans to executes the query

1 table scan
2 index scan

=> in table scan sql server scans whole table to find the desired record
=> in index scan on avg sql server scan only half of the table to find desired record
=> optimizer estimates cost of each plan and selects best plan

execution :-

=> sql server executes the query according to plan selected by optimizer.

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

=> indexes created on columns

1 which are frequently accessed in where clause
2 which are involved in join operation

Types of Indexes :-
---

1 Non Clustered
  simple
  composite
  unique
2 Clustered

05-oct-21

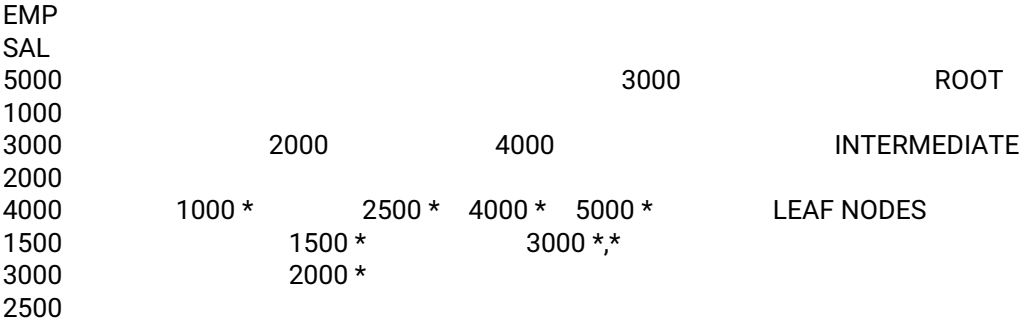
simple non clustered index :-

=> if index created on single column then it is called simple index.

syn :- CREATE INDEX <NAME> ON <TABNAME>(COLNAME)

EX :- CREATE INDEX I1 ON EMP(SAL)

=> after executing above command in db one structure is created called BTREE (balanced binary tree)



select * from emp where sal=3000 (index scan)
select * from emp where sal>=3000 (index scan)
select * from emp where sal<=3000 (index scan)

composite index :-

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

CREATE INDEX I2 ON EMP(DEPTNO,JOB)

=> SQL SERVER uses above index when where condition based on leading column of the index i.e. deptno

SELECT * FROM emp WHERE deptno=20 (INDEX)
SELECT * FROM emp WHERE deptno=20 and job='CLERK' (INDEX)
SELECT * FROM emp WHERE job='CLERK' (TABLE)

unique index :-

=> 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 *
CLARK *

=> sql server uses above index when where condition based on ename column

SELECT * FROM emp WHERE ename = 'BLAKE'

INSERT INTO emp(empno,ename,job,sal) VALUES(444,'BLAKE','CLERK',4000) => ERROR

what are the different methods to enforce uniqueness ?

1 declare primary key / unique constraint
2 create unique index

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

06-oct-21

clustered index :-

=> a non clustered index stores pointers to acutal records where as clustered index stores acutal records

=> in non clustered index order of the records in index and order of the records in table will not be same
where as in clustered index this order will be same.

example :-

```
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(60,'D')

50

30 70

10 A 40 C 60 D 80 B

SELECT * FROM cust => sql server goes to clustered index and reads all the leaf nodes from left to right

10 A
40 C
60 D
80 B

=> only one clustered index is allowed per table

=> sql server implicitly creates a clustered index on primary key

=> difference between non clustered and clustered indexes ?

	non clustered	clustered
1	stores pointers to actural records	stores actual records
2	order of the records in table and order of the records in index will not be same	index and table stored together so order will be same
3	needs extra storage	doesn't need extra storage
4	requires two lookups to fetch record	needs only lookup to fetch record
5	performance is less compared to clustered	gives good performance
6	999 non clustered indexes allowed per table	only one clustered index is allowed per table
7	created explicitly	created implicitly on primary key columns

=> list of indexes created on table ?

sp_helpindex cust

Droping index :-

DROP INDEX EMP.I2

if we drop table what about indexes created on table ?

ans :- indexes are also dropped

SERVER
DB4PM
TABLES
ROWS & COLS
CONSTRAINTS
INDEXES
TRIGGERS
VIEWS

PIVOT operator :-

=> used for cross tabulation or matrix report
=> used to convert rows into columns

```
SELECT *
FROM (SELECT columns
        FROM table
        ) AS <ALIAS>
PIVOT
 (
    aggr-expr for   COLNAME   in (V1,V2,V3,--)
 ) AS <ALIAS>
ORDER BY COL ASC/DESC
```

example 1 :-

	10	20	30
ANALYST	??	??	??
CLERK	??	??	??
MANAGER	??	??	??
SALESMAN	??	??	??

```
SELECT *
FROM (SELECT deptno,job,sal FROM emp) AS E
PIVOT
```

```
  (
    SUM(sal) FOR deptno IN ([10],[20],[30])
  ) AS PIVOT_TBL
 ORDER BY job ASC
```

example 2 :-

```
            1    2    3    4

    1980         ?    ?    ?    ?

    1981         ?    ?    ?    ?

    1982         ?    ?    ?    ?

    1983         ?    ?    ?    ?
```

```
  SELECT *
  FROM (SELECT DATEPART(yy,hiredate) as year,
               DATEPART(qq,hiredate) as qrt,
               empno
       FROM emp) AS E
  PIVOT
    (
      COUNT(empno) FOR qrt IN ([1],[2],[3],[4])
    ) AS PIVOT_TBL
   ORDER BY year ASC
```

Example 3 :- converting rows into columns

```
 STUDENT
 SID     SNAME  SUBJECT      MARKS
  1  A   MAT         90
  1  A   PHY    80
  1  A   CHE    70
  2  B   MAT         80
  2  B   PHY    70
  2  B   CHE    50
```

OUTPUT :-

```
 SID     SNAME  MAT      PHYCHE
 1   A   90   80   70
 2   B   80   70   50
```

```
SELECT *
FROM STUDENT
PIVOT
 (
   SUM(MARKS) FOR SUBJECT IN ([MAT],[PHY],[CHE])
 ) AS PIVOT_TBL
ORDER BY SID ASC
```

08-oct-21

TSQL programming

basic tsql programming
conditional stmts
loop
cursors
error handling
stored procedures
functions
triggers
dynamic sql

Features :-

1 improves performance :-

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

2 supports conditional statements :-

=> supports conditional statements like if-then-else

3 supports loops :-

=> tsql programming supports loops like while loop.

4 supports error handling :-

=> in tsql programming , if any statement causes runtime error then we can handle that error
and we can replace system generatetd message with our own simple and user friendly message

5 supports reusability :-

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

6 supports security :-

=> because tsql programs are stoed in db so only authorized users can executes these programs.

=> TSQL blocks are 2 types

1 anonymous block
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

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 = 1000
SET @s='abc'
SET @d=GETDATE()

PRINT statement :-

=> used to print message or values

PRINT 'hello'
PRINT @x
PRINT @s
PRINT @d

=> write a prog to add two numbers ?

```
DECLARE @a int,@b int,@c int
SET @a=100
SET @b=200
```

```
SET @c=@a+@b
PRINT @c
```

=> write a prog to input date and print day of the week ?

```
DECLARE @d date
SET @d='2022-01-01'
PRINT DATENAME(dw,@d)
```

DB programming with TSQL :-

=> to perform operations over db execute sql commands from tsql program.
=> the following commands can be executed from tsql program

1 DML (insert,update,delete)
2 DQL (select)
3 TCL (commit,rollback,save transaction)

SELECT stmt syntax :-

```
SELECT @var1=col1,
       @var2=col2,
       @var3=col3,
       ---------
FROM tabname
WHERE condition
```

Example 1 :- write a prog to input empno and print name & salary ?

```
DECLARE @eno int,@name varchar(10),@sal money
SET @eno=109
SELECT @name=ename,@sal=sal FROM emp WHERE empno = @eno
PRINT @name + '    ' + CAST(@sal AS VARCHAR)
```

Example 2 :- write a prog to iput empno and print experience ?

```
DECLARE @eno int,@hire date,@expr tinyint
SET @eno=108
SELECT @hire=hiredate FROM emp WHERE empno=@eno
SET @expr = DATEDIFF(yy,@hire,GETDATE())
PRINT CAST(@expr AS VARCHAR) + ' years '
```

09-oct-21

conditional statements :-
---

1 if-else
2 multi if
3 nested if

1 if-else :-

```
 if cond
  begin
       statements
  end
 else
  begin
     statements
  end
```

2 multi if :-

```
 if cond1
  begin
       statements
  end
 else if cond2
  begin
     statements
  end
else if cond3
 begin
    statements
 end
else
  begin
     statements
  end
```

3 nested if :-

```
 if cond
  begin
          if cond
            begin
                 statements
            end
          else
            begin
               statements
            end
  end
else
 begin
    statements
 end
```

Example 1 :-

write a prog to input empno and increment salary 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
```

Example 2 :-

=> write a prog to input empno and increment employee salary as follows ?

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

DECLARE @eno int,@job varchar(10),@pct int
SET @eno=110
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
```

Example 3 :-

```
ACCOUNTS
ACCNO ACTYPE BAL
100      S    10000
101      S    20000
```

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

```
DECLARE @acno int,@type char(1),@amt money,@bal money
SET @acno=100
SET @type='w'
SET @amt=1000
IF @type='W'
  BEGIN
     SELECT @bal=bal FROM accounts WHERE accno=@acno
     IF @amt > @bal
        PRINT 'insuffiient balance'
     ELSE
        UPDATE accounts SET bal=bal-@amt WHERE accno=@acno
  END
ELSE IF @type='D'
     UPDATE accounts SET bal=bal+@amt WHERE accno=@acno
ELSE
     PRINT 'INVALID TRANSACTION'
```

Example 4 :-

write a prog to process money transfer ?

```
DECLARE @sacno int,@tacno int,@amt money,@bal money,@cnt1 int,@cnt2 int
SET @sacno=100
SET @tacno=101
SET @amt=1000
SELECT @bal=bal FROM accounts WHERE accno=@sacno
IF @amt > @bal
      PRINT 'insufficient balance'
ELSE
   BEGIN
        BEGIN TRANSACTION
        UPDATE accounts SET bal=bal-@amt WHERE accno=@sacno
        SET @cnt1=@@ROWCOUNT
        UPDATE accounts SET bal=bal+@amt WHERE accno=@stacno
        SET @cnt2=@@ROWCOUNT
        IF @cnt1=1 and @cnt2=1
            COMMIT
        ELSE
            ROLLBACK
   END
```

Example 5 :-

STUDENT

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

RESULT

SNO	STOT	SAVG	SRES

=> write a prog to input sno and calculate total,avg,result and insert into result table ?

11-oct-21

WHILE loop :-

```
WHILE(cond)
BEGIN
     statements
END
```

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

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

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

```
    SET @x = @x+1
  END
```

=> write a prog to print 2022 calendar ?

```
    2022-01-01      ?
    2022-01-02      ?

    2022-12-31      ?

    DECLARE @d1 date,@d2 date
    SET @d1='2022-01-01'
    SET @d2='2022-12-31'
    WHILE(@d1<=@d2)
    BEGIN
        PRINT CAST(@d1 AS VARCHAR) + '      ' + DATENAME(dw,@d1)
        SET @d1 =   DATEADD(dd,1,@d1)
    END
```

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

```
    DECLARE @d1 date,@d2 date
    SET @d1='2021-10-01'
    SET @d2='2021-10-31'
    WHILE(@d1<=@d2)
    BEGIN
        IF DATENAME(dw,@d1)='SUNDAY'
            PRINT CAST(@d1 AS VARCHAR) + '      ' + DATENAME(dw,@d1)
        SET @d1 = DATEADD(dd,1,@d1)
    END
```

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

```
    input :- SACHIN

    output :-

    S
    A
    C
    H
    I
    N

DECLARE @s varchar(20),@x int=1
SET @s='SACHIN'
WHILE(@x<=LEN(@s))
BEGIN
    PRINT SUBSTRING(@s,@x,1)
    SET @x=@x+1
END
```

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

```
  input :- SACHIN
```

output :-

```
S
SA
SAC
SACH
SACHI
SACHIN
```

```
DECLARE @s varchar(20),@x int=1
SET @s='SACHIN'
WHILE(@x<=LEN(@s))
BEGIN
    PRINT SUBSTRING(@s,1,@x)
    SET @x=@x+1
END
```

=> write a prog to print following pattern ?

```
1
22
333
4444
55555
```

```
DECLARE @x int=1
WHILE(@x<=5)
BEGIN
    PRINT REPLICATE(@x,@x)
    SET @x=@x+1
END
```

=> write a prog to input number and print sum of individual digits ?
=> write a prog to input number and print whether it is prime or not ?
=> write a prog to input string and print reverse ?
=> write a prog to input string and check whether string is palindrome or not ?

12-oct-21

CURSORS :-

=> cursors are used to access row-by-row into tsql program.

=> from tsql program , if submit a query to sql server it goes to db and fetch data from db and copies that
data into temporary memory and using cursor we can give name to that memory and access row-by-row
into tsql program and process the row.

=> follow below steps to use cursor

1 declare cursor
2 open cursor
3 fetch records from cursor
4 close cursor
5 deallocate cursor

Declaring cursor :-
---

syn :- DECLARE <NAME> CURSOR FOR SELECT STATEMENT

ex :- DECLARE C1 CURSOR FOR SELECT * FROM emp

Opening cursor :-
---

syn :- open <cursor-name>
ex :- open c1

1 select stmt associated with cursor submitted to sql server
2 sql server executes the query and data returned by query is copied to temporary memory
3 cursor c1 points to that temporary memory

Fetching records from cursor :-
---

syn :- FETCH NEXT FROM <CURSOR-NAME> INTO <VARIABLES>

Ex :- FETCH NEXT FROM C1 INTO @a,@b,@c,--

=> a fetch statement fetches one row at a time but to process mutliple rows fetch statement should be
executed multiple times , so keep the fetch statement inside a loop.

Closing cursor :-
---

close c1

Deallocate cursor :-

DEALLOCATE C1

@@FETCH_STATUS :-

=> system variable that returns fetch status

0 => fetch successful
-1 => 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
```

Example 2 :- write a prog to calculate total sal without using SUM ?

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

Example 3 :- write a prog to calculae MAX salary ?

Example 4 :- write a prog to calculate MIN salary ?

Example 5 :- write a prog to print employee names as follows ?

smith,allen,ward ,jones,martin,----------

```
DECLARE C1 CURSOR FOR SELECT ename FROM emp
DECLARE @name VARCHAR(10),@s VARCHAR(1000)=' '
OPEN C1
FETCH NEXT FROM C1 INTO @name
WHILE(@@FETCH_STATUS=0)
BEGIN
    SET @s = @s + @name + ','
    FETCH NEXT FROM C1 INTO @name
END
    PRINT LEFT(@s,LEN(@S)-1)
    CLOSE C1
    DEALLOCATE C1
```

STRING_AGG() :-

=> function used to concatenate column values

STRING_AGG(colname,seperator)

```
SELECT STRING_AGG(ename,',') FROM emp
```

=> display dept wise employee names ?

```
SELECT DEPTNO,STRING_AGG(ename,',') AS NAMES
FROM emp
GROUP BY DEPTNO
```

```
10      clark,king,miller
20      ford,adams,scott,smith,jones
30      martin,blake,allen,ward,turner,james
```

Example 3 :-

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

RESULT
SNO     STOT     SAVG     SRES
```

```
DECLARE C1 CURSOR FOR SELECT SNO,S1,S2,S3 FROM STUDENT
DECLARE @SNO INT,@S1 INT,@S2 INT,@S3 INT,@TOTAL INT,@AVG DECIMAL(5,2),@RES
CHAR(4)
OPEN C1
FETCH NEXT FROM C1 INTO @SNO,@S1,@S2,@S3
WHILE(@@FETCH_STATUS=0)
BEGIN
  SET @TOTAL = @S1+@S2+@S3
  SET @AVG = @TOTAL/3
  IF @S1>=35 AND @S2>=35 AND @S3>=35 THEN
     SET @RES='PASS'
  ELSE
     SET @RES='FAIL'
  INSERT INTO RESULT VALUES(@SNO,@TOTAL,@AVG,@RES)
  FETCH NEXT FROM C1 INTO @SNO,@S1,@S2,@S3
   END
   CLOSE C1
   DEALLOCATE C1
```

13-oct-21

SCROLLABLE CURSOR :-

=> by default cursor is called forward only cursor and it supports only forward navigation but doesn't support backward navigation, if cursor declared with SCROLL then it is called scrollable cursor and a scrollable cursor supports both forward and backward navigation.

=> a forward only cursor supports only FETCH NEXT statement but a scrollable cursor support following fetch statements.

```
FETCH FIRST              => fetches first record
FETCH NEXT               => fetches next record
FETCH PRIOR              => fetches previous record
FETCH LAST               => fetches last record
FETCH ABSOLUTE N         => fetches Nth record from first record
FETCH RELATIVE N         => fetches Nth record from current record
```

Example 1 :-

```
DECLARE C1 CURSOR SCROLL FOR SELECT ename FROM emp
DECLARE @name VARCHAR(20)
OPEN C1
FETCH FIRST FROM C1 INTO @name
PRINT @name
FETCH ABSOLUTE 5 FROM C1 INTO @name
PRINT @name
FETCH RELATIVE 5 FROM C1 INTO @name
PRINT @name
FETCH LAST FROM C1 INTO @name
PRINT @name
FETCH PRIOR FROM C1 INTO @name
PRINT @name
CLOSE C1
DEALLOCATE C1
```

Example 2 :- write a prog to print every 5th record 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 C1
```

Example 3 :- write a prog to print names in reverse order ?

```
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
```

STATIC & DYNAMIC :-

---

static :-

=> if cursor is static then if we make changes to base table the changes are not applied to cursor

```
DECLARE C1 CURSOR STATIC FOR SELECT sal FROM emp WHERE empno=100
DECLARE @sal MONEY
```

```
OPEN C1
UPDATE emp SET sal=2000 WHERE empno=100
FETCH NEXT FROM C1 INTO @sal
PRINT @sal
CLOSE C1
DEALLOCATE C1
```

Dynamic :-

=> if cursor is dynamic then if we make change to base table the changes are applied to cursor
=> by default cursor is dynamic

```
DECLARE C1 CURSOR DYNAMIC FOR SELECT sal FROM emp WHERE empno=100
DECLARE @sal MONEY
OPEN C1
UPDATE emp SET sal=3000 WHERE empno=100
FETCH NEXT FROM C1 INTO @sal
PRINT @sal
CLOSE C1
DEALLOCATE C1
```

ERROR HANDLING / EXCEPTION HANDLING :-
---

1 syntax errors
2 logical errors
3 runtime errors

=> errors that are raised during program execution are called runtime errors

example 1 :-

```
declare @a tinyint
set @a=1000
print @a      => runtime error
```

example 2 :-

```
declare    @a tinyint
set @a=100
print @a/0       => runtime error
```

=> in TSQL , if any statement causes runtime error then sql server displays error message and continues
program execution. To replace system generated message with our own message then we need to
handle that runtime error.

=> to handle runtime error we need to include a block called TRY---CATCH block

```
BEGIN TRY
   statements        =>    statements causes exception
END TRY
```

```
    BEGIN CATCH
       statements      =>      statements handles exception
    END CATCH
```

=> in try block if any statement causes runtime error then control is transferred to catch block
and executes the statements in catch block.

Example 1 :-

```
  DECLARE @a tinyint,@b tinyint,@c tinyint
  BEGIN TRY
  SET @a=100
  SET @b=0
  SET @c=@a/@b
  PRINT @c
  END TRY
  BEGIN CATCH
     PRINT 'ERROR'
  END CATCH
```

error handling functions :-
---

1 ERROR_NUMBER() => returns error number
2 ERROR_MESSAGE() => returns error message
3 ERROR_SEVERITY() => returns error severity level
4 ERROR_STAE() => returns error state
5 ERROR_LINE() => returns line number

Example 2 :-

```
  DECLARE @a tinyint,@b tinyint,@c tinyint
  BEGIN TRY
  SET @a=100
  SET @b=0
  SET @c=@a/@b
  PRINT @c
  END TRY
  BEGIN CATCH
      IF ERROR_NUMBER()=220
          PRINT 'value exceeding datatype limit'
      ELSE IF ERROR_NUMBER()=8134
          PRINT 'divisor cannot be zero'
  END
```

18-oct-21

Example 3 :-

```
 CREATE TABLE emp66
 (
    empno int primary key,
    ename varchar(10) not null,
```

```
    sal      money check(sal>=3000)
);
```

write a prog to insert data into emp6 table ?

```
DECLARE @eno int,@name varchar(10),@sal money
BEGIN TRY
SET @eno=100
SET @name='A'
SET @sal=4000
INSERT INTO emp66 VALUES(@eno,@name,@sal)
END TRY
BEGIN CATCH
    IF ERROR_NUMBER()=2627
        PRINT 'empno should not be duplicate'
    ELSE IF ERROR_NUMBER()=515
        PRINT 'name should not be null'
    ELSE IF ERROR_NUMBER()=547
        PRINT 'sal >= 3000'
END CATCH
```

USER DEFINE ERRORS :-

=> errors raised by user are called user defined errors.
=> a user can also raise his own error by using RAISERROR statement

```
    RAISERROR(error message,severity level,state)

    severity level => 1 to 25

    state          => 1 to 255
```

Example 4 :-

```
DECLARE @eno int,@name varchar(10),@sal money,@msg varchar(100)
BEGIN TRY
SET @eno=100
SET @name='A'
SET @sal=4000
INSERT INTO emp66 VALUES(@eno,@name,@sal)
END TRY
BEGIN CATCH
    IF ERROR_NUMBER()=2627
        SET @msg='empno should not be duplicate'
    ELSE IF ERROR_NUMBER()=515
        SET @msg='name should not be null'
    ELSE IF ERROR_NUMBER()=547
        SET @msg='sal >= 3000'
    RAISERROR(@msg,15,1)
END CATCH
```

Example 5 :-

=> write a prog to increment specific employee sal by specific amount and sunday updates are not allowed ?

```
DECLARE @eno int,@amt money
SET @eno=110
SET @amt=2000
IF DATENAME(dw,GETDATE())='MONDAY'
    RAISERROR('monday not allowed',15,1)
ELSE
   UPDATE emp SET sal=sal+@amt WHERE empno=@eno
```

Example 6 :-

```
ACCOUNTS
ACCNO BAL
100      10000
101      20000
```

write a prog to process money transfer ?

```
DECLARE @sacno int,@tacno int,@amt money,@bal money,@cnt1 int,@cnt2 int
SET @sacno=100
SET @tacno=101
SET @amt=1000
SELECT @cnt1=COUNT(*) FROM accounts WHERE accno=@sacno
SELECT @cnt2=COUNT(*) FROM accounts WHERE accno=@tacno
IF @cnt1=0
    RAISERROR('source account does not exists',15,1)
ELSE IF @cnt2=0
    RAISERROR('target account does not exists',15,1)
ELSE
 BEGIN
   SELECT @bal=bal FROM accounts WHERE accno=@sacno
   IF @amt > @bal
     RAISERROR('insufficient balance',15,1)
 ELSE
    BEGIN
        BEGIN TRANSACTION
        UPDATE accounts SET bal=bal-@amt WHERE accno=@sacno
        SET @cnt1=@@ROWCOUNT
        UPDATE accounts SET bal=bal+@amt WHERE accno=@tacno
        SET @cnt2=@@ROWCOUNT
        IF @cnt1=1 and @cnt2=1
          COMMIT
        ELSE
          ROLLBACK
    END
 END
```

display list of errors in sql server ?

```
 SELECT * FROM sys.messages
```

how to add user define error to sys.messages table ?

```
EXECUTE    SP_ADDMESSAGE error number,severity level,error message

EXECUTE SP_ADDMESSAGE 50001,15,'sunday not allowed'
```

=> write a prog to increment specific employee sal by specific amount and sunday updates are not allowed ?

```
DECLARE @eno int,@amt money
SET @eno=110
SET @amt=2000
IF DATENAME(dw,GETDATE())='SUNDAY'
    RAISERROR(50001,15,1)
ELSE
   UPDATE emp SET sal=sal+@amt WHERE empno=@eno
```

NAMED TSQL Blocks :-
---

1 stored procedures
2 stored functions
3 triggers

sub-programs :-
---

1 stored procedures
2 stored functions

Advantages :-
---

1 modular programming
2 reusability
3 security
4 invoked from front-end applications
5 improves performance

stored procedures :-

=> a procedure is a named TSQL block that accepts some input and performs some action on db and may
    or may not returns a value.

=> these programs are called stored procedures because they are stored in db.

=> procedures are created to perform one or more dml operations on db.

syntax :-

```
CREATE OR ALTER PROCEDURE <NAME>
parameters if any
AS
   STATEMENTS
```

parameters :-

=> we can declare parameters and we can pass values to parameters
=> parameters are 2 types

1 INPUT
2 OUTPUT

INPUT :-

=> always recieves value
=> default
=> read only parameter

OUTPUT :-

=> always sends value
=> write only parameter

Example 1 :- create a procedure to increment specific employee sal by specific amount ?

```
CREATE OR ALTER PROCEDURE raise_salary
@eno int,
@amt money
AS
  UPDATE emp SET sal=sal+@amt WHERE empno=@eno
```

commands completed successfully

(compiled + stored in db)

Execution :-
---

syn :- EXECUTE procname parameters

ex :-

method 1 :- (positional notation)

```
EXECUTE raise_salary 100,1000
```

method 2 :- (named notation)

```
EXECUTE raise_salary @eno=100,@amt=1000
```

=> named notation is convienent than positional notation because in named notation values can be passed
to parameters in any order.

Example 2 :-

OUTPUT parameter :-

=> create procedure to increment specific employee sal by specific amount and after increment
send the updated sal to calling program ?

```
CREATE OR ALTER PROCEDURE raise_salary
@eno int,
@amt money,
@newsal money OUTPUT
AS
UPDATE emp SET sal=sal+@amt WHERE empno=@eno
SELECT @newsal=sal FROM emp WHERE empno=@eno
```

Execution :-

```
DECLARE @s money
EXECUTE raise_salary    @eno=100,@amt=1000,@newsal=@s OUTPUT
PRINT @s
```

Example 3 :-

ACCOUNTS
ACCNO BAL
100 10000
101 20000

=> create a procedure for money withdrawl ?

```
CREATE OR ALTER PROCEDURE debit
@acno int,
@amt    money,
@newbal money OUTPUT
AS
  DECLARE @bal money
  SELECT @bal=bal FROM accounts WHERE accno=@acno
  IF @amt > @bal
     RAISERROR('insufficient balance',15,1)
  ELSE
     BEGIN
         UPDATE accounts SET bal=bal-@amt WHERE accno=@acno
         SELECT @newbal=bal FROM accounts WHERE accno=@acno
     END


DECLARE @b money
EXECUTE debit 100,20000,@b OUTPUT
PRINT @b
```

Assignments :-

1 create a procedure for money deposit ?
2 create a procedure for money transfer ?

20-oct-21

Example 4 :-

```
CREATE TABLE emp88
(
   empno    INT PRIMARY KEY,
   ename    VARCHAR(10) NOT NULL,
   sal      MONEY CHECK(sal>=3000),
   dno      INT REFERENCES DEPT(DEPTNO)
)
```

=> create a procedure to insert data into emp88 table ?

```
CREATE OR ALTER PROCEDURE insert_emp88
@eno int,
@name varchar(10),
@sal money,
@dno int,
@msg varchar(100) OUTPUT
AS
   BEGIN TRY
   INSERT INTO emp88 VALUES(@eno,@name,@sal,@dno)
   SET @msg = 'record inserted successfully'
   END TRY
   BEGIN CATCH
   SET @msg = ERROR_MESSAGE()
   END CATCH


DECLARE @s VARCHAR(100)
EXECUTE insert_emp88 100,'A',5000,10,@s OUTPUT
PRINT @s
```

USER DEFINE FUNCTIONS :-

=> when predefine functions not meeting our requirements then we create our own functions called
user define functions.

=> a function is also a named TSQL block that accepts some input peforms some calculation and must
return a value.

=> functions are created

1 for calculations
2 to fetch value from db

=> functions are 2 types

1 scalar valued functions (SVF)
2 table valued functions (TVF)

scalar valued functions :-

=> these functions returns one value

=> return type must be scalar types like int,varchar
=> return expression must be a scalar variable

syn :- CREATE OR ALTER
FUNCTION <NAME>(parameters if any) RETURNS <type>
AS
BEGIN
STATEMENTS
RETURN <expr>
END

Example 1 :-

```
CREATE OR ALTER
    FUNCTION CALC(@a int,@b int,@op char(1)) RETURNS int
AS
BEGIN
   DECLARE @c int
   IF @op='+'
     SET @c=@a+@b
   ELSE IF @op='-'
     SET @c=@a-@b
   ELSE IF @op='*'
     SET @c=@a*@b
   ELSE
     SET @c=@a/@b
   RETURN @c
END
```

Execution :-
---

1 sql commands
2 another tsql program
3 front-end applications

Executing from sql commands :-

```
  SELECT DBO.CALC(10,20,'*')    => 200
```

Executing from another tsql program :-

```
DECLARE @c int
SET @c = DBO.CALC(10,20,'*')
PRINT @c
```

Example 2 :-

PRODUCTS

prodid	pname	price
100	A	2000
101	B	1500
102	C	1000

ORDERS

ordid	prodid	qty
1000	100	2
1000	101	2
1000	102	1
1001	100	1

=> create a function to calculate order amount of particular order ?

input :- ordid = 1000
output :- amount = 8000

```
CREATE OR ALTER FUNCTION getOrdAmt(@d int) RETURNS MONEY
AS
BEGIN
    DECLARE C1 CURSOR FOR SELECT o.prodid,o.qty,p.price
                                    FROM orders o INNER JOIN products p
                                      ON o.prodid = p.prodid
                                    WHERE o.ordid = @d
    DECLARE @pid int,@price money,@qty int,@total money=0
    OPEN C1
    FETCH NEXT FROM C1 INTO @pid,@qty,@price
    WHILE(@@FETCH_STATUS=0)
    BEGIN
       SET @total = @total + (@qty*@price)
       FETCH NEXT FROM C1 INTO @pid,@qty,@price
    END
        CLOSE C1
        DEALLOCATE C1
        RETURN @total
 END
```

execution :-

SELECT getOrdAmt(1000) => 8000

Assignment :-

---

create function to calculate experience of particular employee ?

create function to calculate tax ?

create function to check whether given year leap year or not ?

TABLE VALUED FUNCTIONS :-

=> these functions returns records
=> return type of these functions must be table
=> return expression must be select statement
=> table valued function allows only one statement and that statement must be return statement
=> table valued functions are invoked in FROM clause.

syn :- CREATE OR ALTER FUNCTION <name>(parameters if any) RETURNS TABLE

```
        AS
          RETURN (SELECT STATEMENT)
```

example 1 :- create function that returns list of employees working for specific dept ?

```
  CREATE OR ALTER FUNCTION getEmpList(@d int) RETURNS TABLE
  AS
   RETURN (SELECT * FROM emp WHERE deptno = @d)
```

Execution :-

```
 SELECT * FROM dbo.getEmpList(20)
```

example 2 :- create a function to return top N employees based on sal ?

```
 CREATE OR ALTER FUNCTION getTopNEmpList(@n int) RETURNS TABLE
 AS
   RETURN (SELECT *
             FROM (SELECT empno,ename,sal,
                          dense_rank() over (order by sal desc) as rnk
                     FROM emp) AS E
             WHERE rnk<=@n)


SELECT * FROM dbo.getTopNEmpList(5)
```

Assignment :-

1 create a function to get top n employees in each dept ?

Example 3 :-

=> create function to get latest N employees ?

```
  CREATE OR ALTER FUNCTION getLatestNemp(@n int) RETURNS TABLE
  AS
   RETURN (SELECT *
             FROM (SELECT empno,ename,sal,hiredate,
                          ROW_NUMBER() OVER (ORDER BY hiredate DESC) as rno
                     FROM emp) AS E
           WHERE rno<=@n)


 SELECT * FROM dbo.getLatestNemp(5)
```

Assignment :-

CUSTOMERS
CUSTID NAME   ADDR   DOBPHONE

ACCOUNTS
ACCNO ACTYPE BAL   CUSTID

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

```
CREATE SEQUENCE S1
START WITH 1
INCREMENT BY 1
MAXVALUE 999999
```

=> create procedures & functions to implement following bank transactions ?

1 account opening (PROC)
2 account closing (PROC)
3 money deposit (PROC)
4 money withdrawl (PROC)
5 money transfer (PROC)
6 balance enquiry (SVF)
7 particular customer statement between two given dates (TVF)
8 latest N transactions of particular customer (TVF)

=> difference procedures & functions ?

	procedure	functions
1	may or may not returns a value	must return a value
2	can return multiple values	returns one value
3	returns value using OUTPUT parameter	returns value using return statement
4	created to perform dml operations	created to compute value or to fetch value
5	can execute dml commands	dml commands are not allowed in functions
6	cannot be executed from sql commands	executed from sql commands
7	create procedure to update balance	create function to get balance

=> difference between scalar and table valued functions ?

	scalar	table
1	returns one value	return records
2	return type must be scalar types like int,varchar etc	return type must be table
3	return expression is a scalar variable	returns expression is select statement
4	invoked in select clause	invoked in from clause

Droping :-
---

drop procedure raise_salary

drop function dbo.CALC

drop function dbo.getTopNEmpList

23-oct-21

TRIGGERS :-

=> a trigger is also a named TSQL block like procedures but executed implicitly by sql server whenever
user submits DML/DDL commands to sql server

=> triggers are created

1 to control dmls/ddls
2 to enforce complex rules & validations
3 to audit tables
4 to manage replicas (duplicate copy)
5 to generate values for primary key columns

Syntax :-

```
CREATE OR ALTER TRIGGER <NAME>
ON <TABNAME>
AFTER/INSTEAD OF INSERT,UPDATE,DELETE
AS
BEGIN
   STATEMENTS
END
```

AFTER triggers :-

=> if trigger is after then sql server executes the trigger after executing dml

INSTEAD OF :-

=> if trigger is instead of then sql server executes the trigger instead of executing dml

Example 1 :-

=> create trigger to not to allow dmls on emp table on sunday ?

```
CREATE OR ALTER TRIGGER T1
ON EMP
AFTER INSERT,UPDATE,DELETE
AS
    IF DATENAME(DW,GETDATE())='SUNDAY'
      BEGIN
        ROLLBACK
        RAISERROR('sunday not allowed',15,1)
```

```
        END
```

Example 2 :-

=> create trigger to not to allow dmls on emp table as follows ?

```
  mon - fri    <10am and >4pm
  sat            <10am and >2pm
  sun            ----------
```

```
CREATE OR ALTER TRIGGER T2
ON EMP
AFTER INSERT,UPDATE,DELETE
AS
   IF DATEPART(DW,GETDATE()) BETWEEN 2 AND 6
        AND
        DATEPART(HH,GETDATE()) NOT BETWEEN 10 AND 15
         BEGIN
              ROLLBACK
              RAISERROR('only between 10am and 4pm',15,1)
         END
   ELSE IF DATENAME(DW,GETDATE())='SATURDAY'
              AND
              DATEPART(HH,GETDATE()) NOT BETWEEN 10 AND 13
         BEGIN
             ROLLBACK
             RAISERROR('only between 10am and 2pm',15,1)
         END
  ELSE IF DATENAME(DW,GETDATE())='SUNDAY'
         BEGIN
              ROLLBACK
              RAISERROR('sunday not allowed',15,1)
          END
```

Example 3 :-

create a trigger to not to allow to update empno ?

```
CREATE OR ALTER TRIGGER T3
ON EMP
AFTER UPDATE
AS
     IF UPDATE(empno)
       BEGIN
           ROLLBACK
           RAISERROR('empno cannot be updated',15,1)
       END
```

Magic tables :-

---

1 INSERTED
2 DELETED

=> these tables are created and destroyed implicitly during the trigger execution
=> record user is trying to insert is copied to inserted table.
=> record user is trying to delete is copied to deleted table.

=> record user is trying to update is copied to both inserted & deleted table

INSERT INTO emp(empno,ename,sal) VALUES(100,'A',5000) => INSERTED

EMPNO	ENAME	SAL
100	A	5000

DELETE FROM EMP WHERE EMPNO = 100 => DELETED

EMPNO	ENAME	JOB	SAL
100	SMITH	CLERK	800

EMPNO	SAL
110	800

UPDATE EMP SET SAL=2000 WHERE EMPNO=110 => INSERTED

EMPNO	SAL
110	2000

DELETED

EMPNO	SAL
110	800

25-oct-21

=> create a trigger to not to allow users to decrement salary ?

```
CREATE OR ALTER TRIGGER T4
ON EMP
AFTER UPDATE
AS
   DECLARE @OLDSAL MONEY,@NEWSAL MONEY
   SELECT @OLDSAL=SAL FROM DELETED
   SELECT @NEWSAL=SAL FROM INSERTED
   IF @NEWSAL<@OLDSAL
    BEGIN
        ROLLBACK
        RAISERROR('sal cannot be decremented',15,1)
    END
```

UPDATE EMP SET SAL=1000 WHERE EMPNO=100 => ERROR

=> create trigger to insert details into emp_resign table when employee resigns ?

EMP_RESIGN

EMPNO	ENAME	HIREDATE	DOR

```
CREATE TABLE emp_resign
(
   empno int,
   ename varchar(10),
   hiredate date,
   dor date
)
```

```
CREATE OR ALTER TRIGGER T5
ON EMP
AFTER DELETE
AS
  DECLARE @ENO INT,@NAME VARCHAR(10),@HIRE    DATE
  SELECT @ENO=EMPNO,@NAME=ENAME,@HIRE=HIREDATE FROM DELETED
  INSERT INTO EMP_RESIGN VALUES(@ENO,@NAME,@HIRE,GETDATE())


DELETE FROM EMP WHERE EMPNO=100 => DELETED
                                   EXECUTE COMMAND
                                   EXECUTE TRIGGER
```

=> create trigger to not to allow more than 4 employees in a dept ?

```
EMP33
ENO     DNO
1   10
2   10
3   10
4   10
5   10 => not allowed


CREATE OR ALTER TRIGGER T6
ON EMP33
INSTEAD OF INSERT
AS
   DECLARE @eno int,@dno int
   SELECT @eno=eno,@dno=dno FROM INSERTED
   SELECT @cnt=COUNT(*) FROM emp33 WHERE dno=@dno
   IF @cnt=4
       RAISERROR('max 4 emps per dept',15,1)
   ELSE
       INSERT INTO emp33 VALUES(@eno,@dno)


  insert into emp33 values(1,10)
  insert into emp33 values(2,10)
  insert into emp33 values(3,10)
  insert into emp33 values(4,10)
  insert into emp33 values(5,10)     => ERROR


  Auditing :-
----------

  => triggers are also created for auditing
  => Auditing means capturing changes made to table
  => Auditing means monitoring day-to-day actitivies on tables
  => changes are captured in some tables called audit tables


  EMP_AUDIT
  UNAMEOPERATION     OPTIME      NEW_ENO   NEW_ENAMENEW_SAL          OLD_ENO
          OLD_ENAME OLD_SAL
  dbo     INSERT        ???         100          A          5000          NULL          NULL
```

```
NULL
 dbo     UPDATE     ???      100        A        6000        100     A        5000
 dbo     DELETE     ???      NULL       NULL        NULL        100     A
6000
```

```
 CREATE TABLE emp_audit
 (
     uname          varchar(10),
     operation      varchar(10),
     optime    datetime,
     new_enoint,
     new_ename   varchar(10),
     new_sal  money,
     old_eno  int,
     old_ename    varchar(10),
     old_sal    money
  )
```

create trigger to capture changes made to emp table ?

```
 CREATE OR ALTER TRIGGER T7
 ON EMP
 AFTER INSERT,UPDATE,DELETE
 AS
       DECLARE @oldeno int,@oldename varchar(10),@oldsal money
       DECLARE @neweno int,@newename varchar(10),@newsal money
       DECLARE @cnt1 int,@cnt2 int,@op varchar(10)
       SELECT @oldeno=empno,@oldename=ename,@oldsal=sal FROM DELETED
       SELECT @neweno=empno,@newename=ename,@newsal=sal FROM INSERTED
       SELECT @cnt1=COUNT(*) FROM INSERTED
       SELECT @cnt2=COUNT(*) FROM DELETED
       IF @cnt1=1 AND @cnt2=0
           SET @op='INSERT'
       ELSE IF @cnt1=0 AND @cnt2=1
           SET @op='DELETE'
       ELSE
           SET @op='UPDATE'
       INSERT                                    INTO                                    emp_audit
VALUES(USER_NAME(),@op,GETDATE(),@neweno,@newename,@newsal,
                                                  @oldeno,@oldename,@oldsal)
```

Droping trigger :-

```
    DROP TRIGGER T1
```

Disable & Enable trigger :-

=> if trigger is disable then it exists in db but it will not work till it is enable

```
  DISABLE TRIGGER T2 ON EMP
  ENABLE TRIGGER T2 ON EMP
```

26-oct-21

Dynamic SQL :-

=> SQL commands build at runtime are called dynamic SQL commands

ex :- DROP TABLE emp (static sql)

```
DECLARE @TNAME VARCHAR(100)
SET @TNAME='EMP'
DROP TABLE @TNAME (dynamic sql)
```

=> Dynamic SQL is useful when we don't know tablename and column names until runtime.

=> Dynamic SQL commands are executed by using

1 EXEC statement
2 sp_executesql stored procedure

using EXEC statement :-

=> dynamic sql command that you want to execute should passed as string to EXEC statement

syn :- EXEC (' dynamic sql command ')

Example 1 :- create a procedure to drop table from db ?

```
CREATE OR ALTER PROCEDURE DROP_TABLE
@tname varchar(20)
AS
    EXEC (' DROP TABLE ' + @tname)
```

Example 2 :- create procedure to drop all tables ?

```
CREATE OR ALTER PROCEDURE DROP_ALL_TABLES
AS
  DECLARE C1 CURSOR FOR  SELECT TABLE_NAME
                         FROM INFORMATION_SCHEMA.TABLES
                         WHERE TABLE_TYPE='BASE TABLE'
  DECLARE @TNAME VARCHAR(20)
  OPEN C1
  FETCH NEXT FROM C1 INTO @TNAME
  WHILE(@@FETCH_STATUS=0)
  BEGIN
    EXEC (' DROP TABLE ' + @TNAME)
    FETCH NEXT FROM C1 INTO @TNAME
  END
    CLOSE C1
    DEALLOCATE C1
```

using sp_executesql :-
---

synatax :- SP_EXECUTESQL dynamic sql command,parameters,expressions

=> write a prog to display no of rows in each and every table ?

EMP 14
DEPT 4
CUST 10

```
DECLARE C1 CURSOR FOR  SELECT TABLE_NAME
                       FROM INFORMATION_SCHEMA.TABLES
                       WHERE TABLE_TYPE='BASE TABLE'
DECLARE @TNAME VARCHAR(20),@cmd  VARCHAR(500),@CNT INT
OPEN C1
FETCH NEXT FROM C1 INTO @TNAME
WHILE(@@FETCH_STATUS=0)
BEGIN
   SET @cmd = 'SELECT @CNT=COUNT(*) FROM ' + @TNAME
   EXEC SP_EXECUTESQL @cmd,N'@CNT INT OUTPUT',@CNT=@CNT OUTPUT
   PRINT @TNAME + '     '  + CAST(@CNT AS VARCHAR)
   FETCH NEXT FROM C1 INTO @TNAME
END
   CLOSE C1
   DEALLOCATE C1
```

27=OCT-21

procedure to take backup of all databases ?

```
CREATE OR ALTER PROCEDURE BACKUP_DATABASES
AS
DECLARE C1 CURSOR FOR
                  SELECT name FROM SYS.DATABASES WHERE database_id > 4
DECLARE @DBNAME VARCHAR(20),@FNAME VARCHAR(100)
OPEN C1
FETCH NEXT FROM C1 INTO @DBNAME
WHILE(@@FETCH_STATUS=0)
BEGIN
   SET @FNAME='C:\DATA\' + @DBNAME + CONVERT(VARCHAR,GETDATE(),112) + '.BAK'
   BACKUP DATABASE @DBNAME TO DISK = @FNAME
   FETCH NEXT FROM C1 INTO @DBNAME
END
   CLOSE C1
   DEALLOCATE C1
```

```
execute backup_databases
```

command to copy table from one db to another db :-

```
select * into db6pm.dbo.world_covid_data from  "worldwide covid data"
```

MERGE command :-

=> command used to merge data into a table.
=> command used to manage replicas (duplicate copy)
=> using this command we can apply changes made to one table to another table

scenario :-

27/10

CUSTS
CID NAME CITY
10 A HYD
11 B MUM

=> create replica for custs table ?

SELECT * INTO CUSTT FROM CUSTS

CUSTT
CID NAME CITY
10 A HYD
11 B MUM

28/10

CUSTS
CID NAME CITY
10 A BLR => UPDATED
11 B MUM
12 C DEL => INSERTED

=> use MERGE command to apply changes made to custs to custt

syntax :-

```
MERGE INTO <TARGET-TABLE> <ALIAS>
USING <SOURCE-TABLE> <ALIAS>
ON (CONDITION)
WHEN MATCHED THEN
   UPDATE
WHEN NOT MATCHED THEN
   INSERT                ;
```

Example :-

```
MERGE INTO CUSTT    T
USING CUSTS S
ON (S.CID=T.CID)
WHEN MATCHED THEN
  UPDATE SET T.CITY = S.CITY
WHEN NOT MATCHED THEN
   INSERT VALUES(S.CID,S.CNAME,S.CITY) ;
```