SQL SERVER NOTES

- **Database**: - A database is an organized collection of interrelated data. for example, a university db stores data related to students, courses, faculty etc.
- **Types of Databases**: -
  i) OLTP DB (online transaction processing)
    - Organizations uses OLTP for storing day-to-day transactions.
    -
  ii) OLAP DB (online analytical processing)
    - OLAP for analysis
    - OLAP for analysing business
- **DBMS: - (Database Management System)**

=> It is a software used to create and to manage database.

Evolution of DBMS: -

YEAR	NAME
1960	FMS (File Mgmt. 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
  - according to E.F. CODD in RDBMS in database data must be organized tables i.e., rows & cols

CUSTOMERS

CID NAME AGE => columns/fields/attributes

10 SACHIN 40

11 VIJAY 30

12 RAVI 25 => row/record/tuple

NOTE: Database = collection of tables

Table = collection of rows & cols

Row = collection of field values

Column = collection of values assigned to one field

RDBMS software's: -

SOFTWARE NAME	ORGANIZATION
SQL SERVER	Microsoft
ORACLE	Oracle Corp
DB2	IBM
MYSQL	Oracle Corp
POSTGRESQL	POSTGRESQL forum Dev
RDS	amazon

- sql server is a rdbms product from Microsoft which is used to create and to manage database.
- sql server can be used for both development and administration.

Development	Administration
creating tables	installation of sql server
creating views	creating database
creating synonyms	creating logins
creating sequences	db backup & restore
creating indexes	export & import
creating procedures	db mirroring & replication
creating functions	db upgradation & migration
creating triggers	performance tuning
writing queries	

Versions of sql server: -

VERSION	YEAR	VERSION	YEAR
SQL SERVER 1.1	1991	SQL SERVER 7.0	2000
SQL SERVER 4.2	1993	SQL SERVER 2000	2000
SQL SERVER 6.0	1995	SQL SERVER 2005	2005
SQL SERVER 6.5	1998	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**: - polybase, json, temporal table to save data changes, dynamic data masking and row level security
- **sql server 2017**: - identity cache, New String functions, Automatic Tuning.
- **sql server 2019**: - Read, write, and process big data from Transact-SQL, Easily combine and analyse high-value relational data with high-volume big data, Query external data sources, Store big data in HDFS managed by SQL Server, Query data from multiple external data sources through the cluster.

➢ **CLIENT/SERVER ARCHITECTURE**: -

i) SERVER

- server is a system where sql server software is installed and running.
- inside the server sql server manages databases

ii) CLIENT

- client is also system where users can connect to server, submit requests to server, receives response from server

➢ **How to connect to sql server**: -

To connect to sql server open SSMS and enter following details

- SERVER TYPE: - DB ENGINE
- SERVER NAME: - WINCTRL-F9B3VH5\SQLEXPRESS
- AUTHENTICATION: - SQL SERVER AUTHENTICATION
- LOGIN: - SA (SYSTEM ADMIN)
- PASSWORD: - 123

➢ **CREATING DATABASE IN SQL SERVER**: -

- To create database in sql server in object explorer select

  Database => New Database

- Enter Database Name: - DB6PM
- Click OK

NOTE: a Database is created with following two files

a) DATA FILE (.MDF) (master data file)
b) LOG FILE (.LDF) (log data file)

Data File stores data and log file stores operations

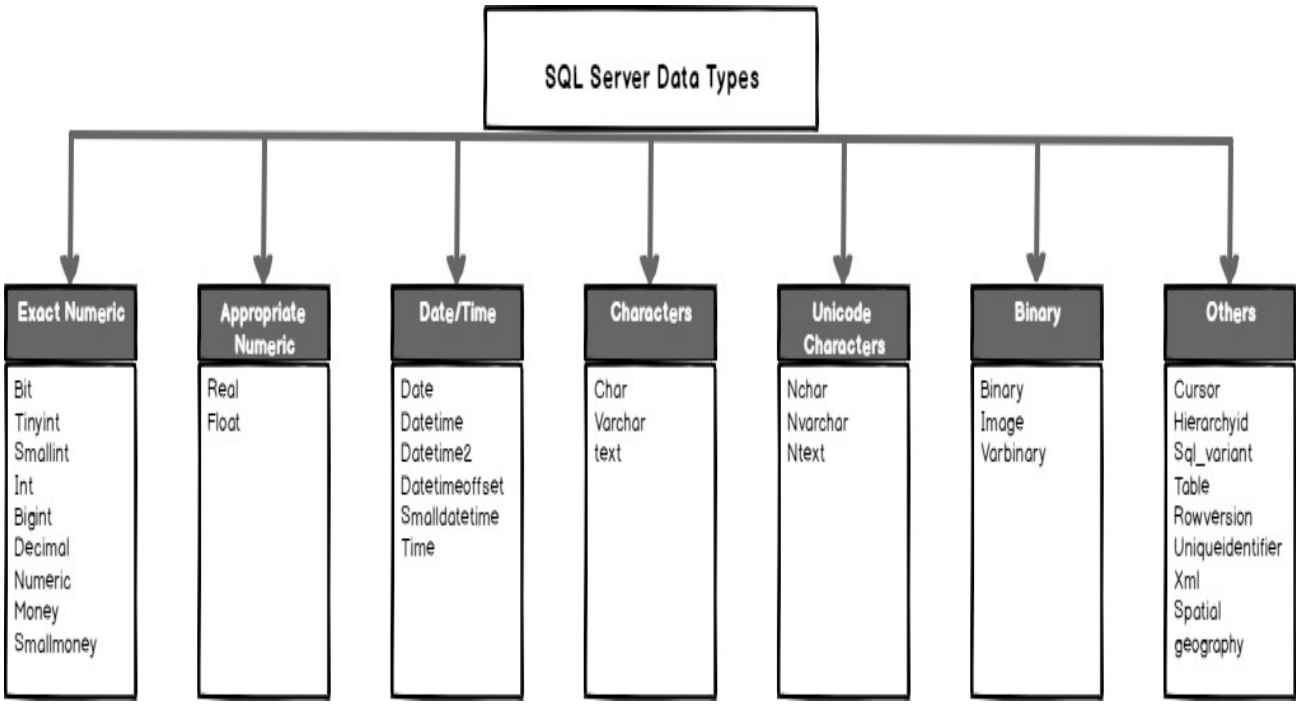
NAME	TYPE	INITIAL SIZE	AUTO GROWTH	PATH
DB6PM	DATA	8MB	64MB	C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS \MSSQL\DATA\
DB6PM_LOG	LOG	8MB	64MB	C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS \MSSQL\DATA\

TSQL (Transact-SQL)

- SQL stands for structured query language.
- This language is used to communicate with sql server.
- User communicates with sql server by sending commands called queries.
- A query is a command/instruction submitted to sql server to perform some operation over db.
- SQL is originally introduced by IBM and initial name of this language was SEQUEL and later it is renamed to SQL.
- SQL is common to all relational databases.
- Based on operations over db sql is categorized into following sublanguages
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
  - DRL/DQL (Data Retrieval Language)
  - TCL (Transaction Control Language)
  - DCL (Data Control Language)

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

- **Datatypes in SQL SERVER**: -
  A datatype specifies type of the data allowed; amount of memory allocated



- **char(size)**: -
  - allows character data up to 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 is not recommended for variable length fields and char is recommended for fixed length fields

ex: - STATE_CODE CHAR (2) AP, AR, AS.

COUNTRY_CODE CHAR (3) IND, USA.

- **VARCHAR**: -
  - allows character data up to 8000 characters
  - recommended for variable length fields

ex: - NAME VARCHAR (10)

Sachin---- released

Ravi------ released

  - in varchar datatype extra bytes are released

- **VARCHAR(MAX)**: -
  - allows character data up to 2GB.
  - using VARCHAR(MAX) we can store large amount of text in db.

NOTE: - char/varchar/varchar(max) allows ascii (American standard code for information interchange) characters

that includes a-z, A-Z,0-9, special characters.

ex: - PANNO CHAR (10)

VEHNO CHAR (10)

EMAILID VARCHAR (30)

- **NCHAR/NVARCHAR/NVARCHAR(MAX): -** (N => National)
  - These types allow Unicode characters (65536 chars) that includes all ascii characters and characters belong to different languages.
  - ascii character occupies 1 byte but a Unicode character occupies 2 bytes.

- **Integer Types**: -
  - ⇨ Integer types allows whole numbers i.e., numbers without decimal part.
  - ⇨ sql server supports 4 integer types

Type	Size	Range
TINYINT	1BYTE	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: - SAL DECIMAL (7,2)

5000

5000.50

50000.50

500000.50 => NOT ACCEPTED

- **CURRENCY TYPES**:
  - ⇨ currency types are used for fields related to money
  - ⇨ sql server supports two currency types

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

ex: - SALARY SMALLMONEY

BALANCE MONEY

- **DATE & TIME**: -

DATATYPE	FORMAT	RANGE
date	YYYY-MM-DD	0001-01-01 through 9999-12-31
time	hh:mi:ss	00:00:00 to 23:59:59
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

  - ⇨ default date format in sql server is yyyy-mm-dd

- ⇨ default time format is hh:mi:ss

ex: - DOB DATE

1995-10-15

LOGIN TIME

10:00:00

TXNDATE DATETIME

2021-08-17 10:00:00

- **BINARY TYPES:** -
  - ⇨ binary types allow binary data that includes audio, video, images.
  - ⇨ sql server supports 3 binary types
    1) **BINARY**
       - ⇨ allows binary data up to 8000 bytes
       - ⇨ recommended for fixed length
       - ⇨ extra bytes are wasted
       - ⇨ Ex: photo binary (8000)
    2) **VARBINARY**
       - ⇨ allows binary data up to 8000 bytes
       - ⇨ recommended for variable length fields
       - ⇨ extra bytes are released
       - ⇨ Ex: photo varbinary (8000)
    3) **VARBINARY(MAX)**
       - ⇨ allows binary data up to 2GB.
       - ⇨ extra bytes are released\
       - ⇨ Ex: photo varbinary(max)

- **CREATING TABLES IN SQL SERVER DB**: -

Syntax:

```
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 up to 128 chars

4) table can have up to 1024 columns
5) table can have unlimited rows

Ex: emp123 valid, 123emp invalid, emp 123 invalid, emp*123 invalid, emp_123 valid

Example: -

a) CREATE TABLE WITH FOLLOWING STRUCTURE?

EMP

EMPID ENAME JOB SAL HIREDATE AGE

Solution: CREATE TABLE emp

(

empid SMALLINT, ename VARCHAR (10),

job VARCHAR (10), sal SMALLMONEY,

hiredate DATE, age TINYINT)

⇨ above command created table structure/definition/metadata that includes columns, datatype and size.

❖ **SP_HELP**: - (SP => stored procedure)
- command used to see the structure of the table.

❖ **INSERTING DATA INTO TABLE**: -
- "INSERT" command is used to insert data into table
- "INSERT" command creates new row in table.
- using INSERT command, we can insert
  1) single row
  2) multiple rows

INSERTING SINGLE ROW: -

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

ex: -

INSERT INTO emp VALUES (100,'sachin','clerk',4000,'2021-08-18',45)

INSERT INTO emp VALUES (101,'vijay','analyst',8000, getdate (),30)

INSERTING MULTIPLE ROWS: -

INSERT INTO emp VALUES (102,'ravi','manager',10000,'2020-05-10',28),

(103,'kumar','clerk',5000,'2019-10-05',30),

(104,'satish','analyst',9000, getdate(),37)

INSERTING NULLS: -

- a null means blanks or empty
- it is not equal to 0 or space
- nulls can be inserted in two ways

method 1: -

```
INSERT INTO emp VALUES (105,'ajay', NULL, NULL, getdate(),32)
```

method 2: -

```
INSERT INTO emp (empid, ename, sal, hiredate) VALUES (106,'vinod',12000,'2020-01-01')
```

- remaining fields job, age filled with NULLs.

❖ **Displaying Data**: -

- "SELECT" command is used to display data from table.
- using SELECT command we can display all rows/cols or specific rows/cols

syntax: - SELECT columns/* FROM tabname

SQL = ENGLISH

QUERIES = SENTENCES

CLAUSES = WORDS

FROM clause => specify tablename

SELECT clause => specify column names

* => all columns

- ⇨ display all the data from emp table? SELECT * FROM emp
- ⇨ display employee names and salaries? SELECT ename, sal FROM emp
- ⇨ display employee names, job, age? SELECT ename, job, age FROM emp

Operators in SQL SERVER: -

1) Arithmetic Operators => + - * / %
2) Relational Operators => > >= < <= = <> or! =
3) Logical Operators => AND OR NOT
4) Special Operators => BETWEEN, IN, LIKE, IS, ANY, ALL, EXISTS, PIVOT.
5) Set Operators => UNION, UNION ALL, INTERSECT, EXCEPT

WHERE clause: -

- where clause is used to get specific row/rows from table based on a condition.

Syntax: - SELECT columns

FROM tabname

WHERE condition

condition: -

COLNAME OP VALUE

- OP must be any relational operator like > >= < <= = <>
- if condition = true row is selected
- if condition = false row is not selected
  - ⇨ display employee details whose empid=103?

    SELECT * FROM emp WHERE empid=103
  - ⇨ display employee details whose name = vinod ?

    SELECT * FROM emp WHERE ename='vinod'
  - ⇨ display employees earning more than 5000?

    SELECT * FROM emp WHERE sal>5000
  - ⇨ 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 employee list working as clerk,analyst ?

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

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

=> display employee list where empid=100,103,105 ?

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

=> display employees working as clerk and age > 35 ?

SELECT * FROM emp WHERE job='clerk' AND age>35

=> display employees age between 30 and 40 ?

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

=> display employees joined in 2020 year ?

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

scenario :-

STUDENTS

SNO SNAME S1 S2 S3

1 A 80 90 70

2 B 60 30 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. i.e. "=" comparision with multiple values.

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

=> used BETWEEN operator for range comparision.

WHERE COLNAME BETWEEN V1 AND V2 (COL>=V1 AND COL<=V2)

=> employees earning between 5000 and 10000 ?

SELECT * FROM emp WHERE sal BETWEEN 5000 AND 10000

=> employees joined in 2020 year ?

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

=> employee list age not between 30 and 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 (SAL>=5000 AND SAL<=10000)

SELECT * FROM EMP WHERE SAL BETWEEN 10000 AND 5000 (SAL>=10000 AND SAL<=5000)

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

21-aug-21

LIKE operator :-

=> use LIKE operator when comparision with pattern

syn :- WHERE COLNAME LIKE 'PATTERN'

WHERE COLNAME NOT LIKE 'PATTERN'

=> pattern consists of alphabets,digits,wildcard characters.

wildcard characters :-

% => 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 'r' ?

SELECT * FROM emp WHERE ename LIKE '%r'

=> name contains 'a' ?

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

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

SELECT * FROM emp WHERE ename LIKE '_a%'

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

SELECT * FROM emp WHERE ename LIKE '%a__'

=> name starts with 'a' and ends with 'y' ?

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

=> name contains 4 chars ?

SELECT * FROM emp WHERE ename LIKE '____'

=> employees joined in october month ? yyyy-mm-dd

SELECT * FROM emp WHERE hiredate LIKE '_____10___'

SELECT * FROM emp WHERE hiredate LIKE '_____10%'

=> employees joined in 2020 year ?

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

Question :-

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

A ERROR

B RETURNS clerk,manager

C RETURNS only clerk

D none

ans :- C

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

ans :- B

IS operator :-

=> use IS operator when comparision with NULL /NOT NULL

```
WHERE COL IS NULL
WHERE COL IS NOT NULL
```

=> display employees not earning salary ?

SELECT * FROM emp WHERE sal IS NULL

=> display employees earning salary ?

SELECT * FROM emp WHERE sal IS NOT NULL

summary :-

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

23-AUG-21

Assignment :-

CUST

CID NAMEGENDER CITY AGE

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

ALIAS :-

=> alias means another name or alternative name

=> aliases are used to change column heading

syn :- colname/expr [AS] alias

=> display ENAME,ANNUAL SALARY ?

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

```
SELECT ename,sal*12 as "annual salary"
FROM emp
```

=> display ENAME,SAL,HRA,DA,TAX,TOTSAL ?

HRA = house rent allowance = 20% on sal

DA = dearness allowance = 30% on sal

TAX = 10% on sal

TOTSAL = SAL + HRA + DA - TAX

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

ORDER BY clause :-

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

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

=> default order is ASC for DESC order use DESC option.

=> arrange employee list name wise asc order ?

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

=> arrange employee list sal wise desc order ?

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

=> arrange employee list hiredate wise asc and employee joined on same date should be sorted age wise asc ?

```
SELECT *
FROM emp
ORDER BY hiredate ASC ,age ASC
```

=> arrange employee list hiredate wise asc and employee joined on same date should be sorted sal desc order ?

```
SELECT *
FROM emp
ORDER BY hiredate ASC ,sal DESC
```

SCENARIO :-

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 ,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 the output ?

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

NOTE :-

1

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

```
SELECT *
FROM emp
ORDER BY 4 DESC ;
```

=> above query sorts data based on 4th column in emp table i.e. sal

2 ORDER BY number is not according to table it should be according to select list

```
SELECT empid,ename,sal
FROM emp
ORDER BY 4 ASC ;  => ERROR
```

=> to sort based on sal

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

ORDER BY 3 ASC ;

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

```
SELECT *
FROM emp
WHERE job IN ('clerk','manager')
ORDER BY sal ASC
```

24-AUG-21

TOP clause :-

=> used to select Top N rows

=> used to limit no of rows return by select statement

SYN :- SELECT TOP <number> columns FROM tabname

=> display first 3 rows from emp table ?

```
SELECT TOP 3 *
FROM emp
```

=> display top 3 employees based on sal ?

```
SELECT TOP 3 *
FROM emp
ORDER BY sal DESC
```

=> display top 3 max salaries ?

```
SELECT TOP 3 sal
FROM emp
ORDER BY sal DESC
```

=> display top 3 employees based on experience ?

```
SELECT TOP 3 *
FROM emp
ORDER BY hiredate ASC
```

DML(Data Manipulation Language) commands :-

---

insert
update
delete
merge

=> all DML commands acts on table data.
=> in sql server by default operations are auto committed.
=> to stop this auto commit execute the following command

```
SET IMPLICIT_TRANSACTIONS ON
```

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

UPDATE command :-

=> update command is used to modify the data in a table.

=> using update command we can update all rows or specific rows

=> using update command we can update single column or multiple columns

syn :-

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

Ex :-

1 update all the employees comm with 500 ?

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

2 update employees comm with 500 whose comm=null ?

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

25-aug-21

3 update employees comm to null whose comm <> null ?

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

NULL assignment use = operator

NULL comparision use IS operator

4 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%'
```

5 transfer all employees from 10th dept to 20th dept ?

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

DELETE command :-

=> command used to delete row/rows from table

=> using delete command we can delete all rows or specific rows

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

ex :- delete all rows from table ?

```
DELETE FROM emp
```

delete employee row whose empno=7369 ?

```
DELETE FROM emp WHERE empno=7369
```

delete employees joined in 1980 ?

DELETE FROM emp WHERE hiredate LIKE '1980%'

delete employees earning between 1000 and 2000 ?

DELETE FROM emp WHERE sal BETWEEN 1000 AND 2000

delete employees working as clerk,manager ?

DELETE FROM emp WHERE job IN ('clerk','manager')

DDL(DATA DEFINITION LANGUAGE) command :-

CREATE

ALTER

DROP

TRUNCATE

=> all DDL commands acts on table structure or table definition

=> by default all DDL commands are auto committed.

=> to stop auto commit then execute the following command

SET IMPLICIT_TRANSACTIONS ON

27-aug-21

ALTER command :-

=> command used to modify the structure

=> using ALTER command we can

1 add columns

2 drop columns

3 modify column

incr/decr field size

changing datatype

Adding column :-

=> add column gender to emp table ?

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

=> after adding by default the column is filled with nulls , use update command to insert data into this column

```
UPDATE emp SET gender='M' WHERE empno=7499
```

Droping column :-

=> drop column gender from emp ?

```
ALTER TABLE emp
```

DROP COLUMN gender

Modifying column :-

=> 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.
```

=> modify column sal datatype to money ?

```
ALTER TABLE emp
   ALTER COLUMN sal MONEY
```

=> modify column empno datatype to TINYINT ?

ALTER TABLE emp

ALTER COLUMN empno TINYINT => ERROR because existing values in empno column

not in TINYINT range.

Drop command :-

=> drops table from database.

=> drops table structure along with data.

syn :- DROP TABLE <tabname>

ex :- DROP TABLE emp

TRUNCATE command :-

=> deletes all the data from table but keeps structure

=> will empty the table

=> releases all the pages (memory) allocated for table.

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 and specific rows	can delete only all rows but cannot delete specific rows
3	where cond can be used with delete	where cond cannot be used with truncate
4	deletes row-by-row	deletes all rows at a time
5	slower	faster
6	will not release memory	relases memory
7	used by developers	used by DBAs
8	will not reset identity	will reset identity

SP_RENAME :-

=> command used to change tablename or column name

SP_RENAME <oldname>,<newname>

=> rename table student to stud ?

SP_RENAME 'STUDENT','STUD'

=> rename column comm to bonus in emp table ?

SP_RENAME 'EMP.COMM','BONUS'

28-AUG-21

creating new table from existing table (replica) :-

---

syn :- SELECT <cols>/* INTO <new-tabname> FROM <old-tabname>

ex :- SELECT * INTO emp10 FROM emp

=> above command creates a new from query output

=> creates a new table with name EMP10 and copies rows & columns and return by query

copying specific rows & cols :-

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

copy only structure(columns) but not data(rows) :-

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

copying data from one table to another table :-

---

syn :-

```
INSERT INTO  <target-table>
SELECT columns FROM <source-table>
```

ex :- copy data from EMP to EMP12

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

how to change column positions :-

---

step 1 :- create a new temporary table

select empno,ename,job,sal,bonus,deptno,mgr,hiredate INTO temp from emp12

step 2 :- drop original table emp12

drop table emp12

step 3 :- rename table temp to emp12

sp_rename 'temp','emp12'

IDENTITY :-

=> used to generate sequence numbers
=> used to auto increment column values
=> identity is declared while creating table
=> identity is declared for integer columns

syntax :- 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
```

30-aug-21

DELETE VS TRUNCATE :-

```
SELECT * FROM cust                    SELECT * FROM cust
```

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

DELETE FROM CUST

CID	CNAME
105	K

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

TRUNCATE TABLE CUST

CID	CNAME
100	K

how to reset identity manually :-

DBCC CHECKIDENT (tablename,reseed,value)

ex :- DBCC CHECKIDENT('CUST',RESEED,99)

DBCC => DB consistency check

how to propvide explicit value for identity column :-

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

INSERT INTO cust(cid,cname) values(110,'K') => ERROR

=> execute the following command to provide explicit value for identity column

SET IDENTITY_INSERT CUST ON

INSERT INTO cust(cid,cname) values(110,'K') => 1 row affected

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 date & time and milliseconds

SELECT GETDATE() => 2021-08-30 18:32:58.743

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

DATEPART(interval,DATE)

SELECT DATEPART(yy,GETDATE()) => 2021

mm => 08

dd => 30

dw => 02 (day of the week)

01 sunday

02 monday

07 saturday

dayofyear => 242 (day of the year)

hh => hour part

mi => minutes

ss => seconds

qq => 3 (quarter)

1 jan-mar

2 apr-jun

3 jul-sep

4 oct-dec

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

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

=> display employees joined in leap year ?

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

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

```
SELECT * FROM emp WHERE DATEPART(mm,hiredate) IN (1,4,12)
```

=> 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
```

31-aug-21

DATENAME() :-

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

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

	MM	DW
DATEPART	08	03
DATENAME	August	Tuesday

=> 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() :-

=> returns difference between two dates in given interval

DATEDIFF(interval,start date,end date)

```
SELECT DATEDIFF(yy,'2020-08-31',GETDATE())  => 1
SELECT DATEDIFF(mm,'2020-08-31',GETDATE())  => 12
SELECT DATEDIFF(dd,'2020-08-31',GETDATE())  => 365
```

=> display ENAME,EXPERIENCE in year ?

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

=> display employees having more than 40 years of experience ?

```
SELECT ename,DATEDIFF(yy,hiredate,GETDATE()) as experience
FROM emp
WHERE DATEDIFF(yy,hiredate,getdate()) > 40
```

=> display ENAME , EXPERIENCE ?

M YEARS N MONTHS

EXPERIENCE = 40 MONTHS = 3 YEARS 4 MONTHS

YEARS = MONTHS/12 = 40/12 = 3

MONTHS = MONTHS % 12 = 40 % 12 = 4

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

DATEADD() :-

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

DATEADD(interval,int,date)

SELECT DATEADD(dd,10,GETDATE()) => 2021-09-10 18:35:02.780

SELECT DATEADD(MM,1,GETDATE()) => 202-09-30 18:36:06.207

SELECT DATEADD(YY,1,GETDATE()) => 2022-08-31 18:36:06.207

scenario :-

GOLD_RATES

DATEID	RATE
2015-01-01	?
2015-01-02	?

2021-08-31 ?

=> display today's gold rate ?

=> display yesterday's gold rate ?

=> dsiplay last month same day gold rate ?

=> display last year same day gold rate ?

EOMONTH() :-

=> returns month last day

EOMONTH(date,int)

SELECT EOMONTH(GETDATE(),0) => 2021-08-31

SELECT EOMONTH(GETDATE(),1) => 2021-09-30

SELECT EOMONTH(GETDATE(),-1) => 2021-07-31

Assignment :-

1 display first day of the current month ?

2 display next month first day ?

3 display current year first day ?

4 display next year first day ?

01-SEP-21

string functions :-

UPPER() :- converts string to uppercase

UPPER(string)

SELECT UPPER('hello') => HELLO

LOWER() :- converts string to lowercase

SELECT LOWER('HELLO') => hello

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

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

=> convert names to lowercase in table ?

UPDATE emp SET ename = LOWER(ename)

LEN() :-

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

LEN(string)

SELECT LEN('hello') => 5

=> display employee list name contains 5 chars ?

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

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

LEFT & RIGHT :-

=> used to extract part of the string

LEFT(string,len) => returns chars starting from left side

RIGHT(string,len) => returns chars starting from right side

```
SELECT LEFT('hello welcome',5)   => hello
SELECT RIGHT('hello welcome',7)  => welcome
```

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

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

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

scenario :-

=> generate emailids for employees ?

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

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

=> store emailids in db ?

step 1 :- add emailid column to emp table

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

step 2 :- fill the column with emailids

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

SUBSTRING() :-

=> used to extract part of the string starting from specific position.

```
SUBSTRING(string,start,len)

SELECT SUBSTRING('hello welcome',7,4)    =>  welc
SELECT SUBSTRING('hello welcome',10,3)   =>  com
```

REPLICATE() :-

=> repeats given char 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

12345678932

your a/c no XXXX8932 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','elo','abc') => hello

SELECT REPLACE('hello','ell','') => ho

02-SEP-21

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

TRANSLATE(str1,str2,str3)

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

e => a

l => b

o => c

=> translate function can be used to encrypt data i.e. changing plain text to cipher text.

=> display ENAME SAL ?

SELECT ename,

TRANSLATE(sal,'0123456789.','$B*T%@p^#K$') as sal

FROM emp

2975.00 => *K^@$$$

Assignment :-

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

output :- hello

CHARINDEX :-

=> returns position of a character in a string.

CHARINDEX(char,string,[start])

SELECT CHARINDEX('o','hello welcome') => 5

SELECT CHARINDEX('x','hello welcome') => 0

SELECT CHARINDEX('o','hello welcome',6) => 11

Assignment :-

CUST

CID	CNAME
10	sachin tendulkar
11	virat kohli

display CID FNAME LNAME ?

10 sachin tendulkar

MATEHAMTICAL 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 of a number

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(10-10) => 0

ROUND() :- used to round number to integer or to decimal places based on average

37.58948383 => 37
37.58
37.5894

syn :- ROUND(number,decimal places)

SELECT ROUND(37.58948,0) => 38

37----------37.5----------38

number < avg => rounded to lowest

number >= avg => rounded to highest

SELECT ROUND(37.38948,0) => 37

SELECT ROUND(37.58948,2) => 37.59

SELECT ROUND(37.58348,2) => 37.58

SELECT ROUND(381,-2) => 400

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

SELECT ROUND(323,-2) => 300

SELECT ROUND(381,-1) => 380

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

SELECT ROUND(381,-3) => 0

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

03-sep-21

CEILING() :- rounds number always to highest

CEILING(number)

CEILING(3.1) => 4

FLOOR() :- rounds number always to lowest

FLOOR(number)

FLOOR(3.9) => 3

conversion functions :-

=> these functions are used to convert one datatype to another datatype

1 CAST

2 CONVERT

CAST :-

CAST(source-expr as target-type)

SELECT CAST(10.5 AS INT) => 10

SELECT 10/4 => 2

SELECT CAST(10 AS FLOAT)/4 => 2.5

=> display smith earns 800
allen earns 1600 ?

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

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

NOTE :- in concatenation all expressions must be character type

CONVERT() :- used to convert one datatype to another datatype

CONVERT(target-type,source-expr)

SELECT CONVERT(INT,10.5)

diff b/w CAST & CONVERT ?

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

Displying 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	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 SAL HIREDATE ? display hiredates in dd.mm.yyyy format ?

SELECT empno,ename,sal,CONVERT(varchar,hiredate,104) as hiredate FROM emp

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

04-sep-21

Analytical functions :-

=> used for data analysis

=> used to do Top N analysis

RANK & DENSE_RANK functions :-

=> both functions are used to calculate 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
```

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

what is diff b/w rank & dense_rank ?

1 rank function generates gaps but dense_rank will not generate gaps.

2 in rank function ranks may not be in sequence but in dense_rank function ranks will be always in sequence.

SAL	RNK	DRNK
5000	1	1
4000	2	2
3000	3	3
3000	3	3
3000	3	3
2000	6	4
2000	6	4
1000	8	5

=> 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
```

PARTITION BY clause :-

=> partition by clause is 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 rank/dense_rank function on each dept instead of applying it on whole table.

=> display ranks of the employees 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
```

10	7839	king	5000.00	1
10	7782	clark	2450.00	2
10	7934	miller	1300.00	3
20	7902	ford	3000.00	1
20	7788	scott	3000.00	1
20	7566	jones	2975.00	2
20	7876	adams	1100.00	3
20	7369	smith	800.00	4
30	7698	blake	2850.00	1
30	7499	allen	1600.00	2
30	7844	turner	1500.00	3
30	7521	ward	1250.00	4
30	7654	martin	1250.00	4
30	7900	james	950.00	5

06-sep-21

ROW_NUMBER() :-

=> returns record numbers for the records after sorting

syn :- ROW_NUMBER() OVER ([PARTITION BY <col>] ORDER BY col ASC/DESC)

```
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
7698	blake	2850.00	6
7782	clark	2450.00	7
7788	scott	3000.00	8
7839	king	5000.00	9
7844	turner	1500.00	10
7876	adams	1100.00	11
7900	james	950.00	12
7902	ford	3000.00	13
7934	miller	1300.00	14

SPECIAL FUNCTIONS :-

ISNULL() :-

=> used to convert nulls values

syn :- ISNULL(arg1,arg2)

if arg1 = null returns arg2

if arg1 <> null returns arg1 only

SELECT ISNULL(100,200) => 100

SELECT ISNULL(NULL,300) => 300

=> 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

=> Display ENAME,SAL,COMM ? if comm = null display N/A ?

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

Aggregate functions :-

=> 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 = >1983-01-12

MIN() :- returns minimum 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 with thousand seperator ?

SELECT CONVERT(VARCHAR,ROUND(SUM(sal),-2),1) FROM emp => 29,000

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

AVG() :- returns average value

AVG(arg)

SELECT AVG(sal) FROM emp => 2073.2142

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

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

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

COUNT(arg)

SELECT COUNT(empno) FROM emp => 14

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

SELECT COUNT(DISTINCT deptno) FROM emp => 3

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

SELECT COUNT(*) FROM emp => 14

=> difference between count & count(*) ?

count function ignores nulls but count(*) includes nulls

T1
F1
10
NULL
20
NULL
30

COUNT(F1) => 3

COUNT(*) => 5

07-SEP-21

=> display no of employees joined in 1981 year ?

```
SELECT COUNT(*) FROM emp WHERE hiredate LIKE '1981%'
```

=> display no of employees joined on sunday ?

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

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

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

NOTE :- aggregate functions not allowed in where clause and they are allowed only in select,having clauses.

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

summary :-

DATE :- datepart,datename,datediff,dateadd,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_ran,row_number

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

Assignment :-

---

display first day of the month ?

SELECT DATEADD(dd,1,EOMONTH(getdate(),-1))

GROUP BY clause :-

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

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	10
2	B	6000	20
3	C	4000	30

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

DEPTNO	
10	12000
20	14000

4 D 8000 20 30 4000

5 E 7000 10

detailed data summarized data

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

syn :-

```
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 salary ?

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

FROM emp :-

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	10
2	B	6000	20
3	C	4000	30
4	D	8000	20
5	E	7000	10

GROUP BY deptno :-

10

1 A 5000

5 E 7000

20

2 B 6000

4 D 8000

30

3 C 4000

SELECT deptno,SUM(sal) :-

---

10 12000

20 14000

30 4000

08-sep-21

=> 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)
```

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

NOTE :- column alias cannot be referenced in GROUP BY because GROUP BY clause is executed before SELECT.

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

```
SELECT DATEPART(qq,hiredate) as qer,COUNT(*) as cnt
FROM emp
```

```
WHERE datepart(yy,hiredate)=1981
GROUP BY DATEPART(qq,hiredate)
```

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

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

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

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

FROM emp :-

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	10
2	B	6000	20
3	C	4000	30
4	D	8000	20
5	E	7000	10

GROUP BY deptno :-

10

1 A 5000

5 E 7000

20

2 B 6000

4 D 8000

30

3 C 4000

HAVING COUNT(*) > 1 :-

10

1 A 5000

5 E 7000

20

2 B 6000

4 D 8000

SELECT deptno,count(*) :-

---

10 2

20 2

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

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

WHERE VS HAVING :-

	WHERE	HAVING
1	filters rows	filters 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
4	we can apply where condition without group by	for having condition group by is compulsory

09-sep-21

Grouping based on multiple columns :-

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

```
SELECT deptno,job,SUM(sal) as totsal
FROM emp
GROUP BY deptno,job
ORDER BY 1 ASC,2 ASC
```

10	CLERK	1300
	MANAGER	2450
	PRESIDENT	5000
20	ANALYST	6000
	CLERK	1900
	MANAGER	2975
30	CLERK	950
	MANAGER	2850
	SALESMAN	5600

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

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

ORDER BY 1 ASC,2 ASC

ROLLUP & CUBE :-

=> ROLLUP & CUBE are used to display subtotals and grand total

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

ROLLUP :-

=> rollup calculates subtotals for each group and also calculates grand total

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

10	CLERK	1300.00
10	MANAGER	2450.00
10	PRESIDENT	5000.00
10	NULL	8750.00 => subtotal
20	ANALYST	6000.00
20	CLERK	1900.00
20	MANAGER	2975.00
20	NULL	10875.00 => subtotal
30	CLERK	950.00
30	MANAGER	2850.00

```
30      SALESMAN  5600.00
30      NULL      9400.00  => subtotal
NULL    NULL      29025.00 => grand total
```

CUBE :-

=> CUBE displays subtotals for each group by column (i.e. deptno wise and job wise) and also displays grand total.

```
SELECT deptno,job,SUM(sal) as totsal
FROM emp
GROUP BY CUBE(deptno,job)
ORDER BY ISNULL(deptno,99) ASC,ISNULL(job,'Z') ASC
```

```
10      CLERK      1300.00
10      MANAGER    2450.00
10      PRESIDENT  5000.00
10      NULL       8750.00  => dept subtoal
20      ANALYST    6000.00
20      CLERK      1900.00
20      MANAGER    2975.00
20      NULL       10875.00  => dept subtotal
30      CLERK      950.00
30      MANAGER    2850.00
30      SALESMAN   5600.00
30      NULL       9400.00 => dept subtotal
NULL    ANALYST    6000.00 => job subtotal
NULL    CLERK      4150.00 => job subtotal
```

NULL MANAGER 8275.00 => job subtotal

NULL PRESIDENT 5000.00 => job subtotal

NULL SALESMAN 5600.00 => job subtotal

NULL NULL 29025.00 => grand total

11-SEP-21

GROUPING_ID() :-

=> grouping_id functions accepts group by columns and returns subtotal belongs to which group by column

GROUPING_ID(deptno,job)

1 => if subtotal belongs to dept

2 => if subtotal belongs to job

3 => grand total

```
SELECT deptno,job,SUM(sal) as totsal,
       GROUPING_ID(deptno,job) as subtotal
FROM emp
GROUP BY CUBE(deptno,job)
ORDER BY ISNULL(deptno,99) ASC,ISNULL(job,'Z') ASC
```

10 CLERK 1300.00 0

10 MANAGER 2450.00 0

10 PRESIDENT 5000.00 0

10 NULL 8750.00 1

20	ANALYST	6000.00		0
20	CLERK	1900.00		0
20	MANAGER	2975.00		0
20	NULL	10875.00	1	
30	CLERK	950.00	0	
30	MANAGER	2850.00		0
30	SALESMAN	5600.00	0	
30	NULL	9400.00	1	
NULL	ANALYST	6000.00		2
NULL	CLERK	4150.00		2
NULL	MANAGER	8275.00		2
NULL	PRESIDENT	5000.00	2	
NULL	SALESMAN	5600.00	2	
NULL	NULL	29025.00	3	

CASE statement :-

=> CASE statements are used to implement if-then-else

=> 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 as job
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*1.1
        when 20 then sal*1.15
        when 30 then sal*1.2
        else sal*1.05
        end
```

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
```

=> display empno,ename,sal,salrange ?

```
            if sal>3000 display Hisal
              sal<3000 display Losal
                =3000 display Avgsal
SELECT empno,ename,sal,
      CASE
      WHEN sal>3000 THEN 'Hisal'
      WHEN sal<3000 THEN 'Losal'
      ELSE 'Avgsal'
```

```
    END as salrange
FROM emp


SELECT deptno,job,SUM(sal) as totsal,
      CASE GROUPING_ID(deptno,job)
                  WHEN 1 THEN 'Dept Subtotal'
                  WHEN 2 THEN 'Job Subtotal'
                  WHEN 3 THEN 'Grand total'
                  END as subtotal
FROM emp
GROUP BY CUBE(deptno,job)
ORDER BY ISNULL(deptno,99) ASC,ISNULL(job,'Z') ASC
```

10	CLERK	1430.00	NULL
10	MANAGER	2695.00	NULL
10	PRESIDENT	5500.00	NULL
10	NULL	9625.00	Dept Subtotal
20	ANALYST	6900.00	NULL
20	CLERK	2185.00	NULL
20	MANAGER	3421.25	NULL
20	NULL	12506.25	Dept Subtotal
30	CLERK	1140.00	NULL
30	MANAGER	3420.00	NULL
30	SALESMAN	6720.00	NULL
30	NULL	11280.00	Dept Subtotal
NULL	ANALYST	6900.00	Job Subtotal
NULL	CLERK	4755.00	Job Subtotal
NULL	MANAGER	9536.25	job Subtotal
NULL	PRESIDENT	5500.00	Job Subtotal

NULL SALESMAN 6720.00 Job Subtotal

NULL NULL 33411.25 Grand total

Grouping based on range :-

=> display no of employees for each salary range ?

```
 SELECT CASE
       WHEN SAL BETWEEN 0 AND 2000 THEN '0-2000'
          WHEN SAL BETWEEN 2001 AND 4000 THEN '2001-4000'
          WHEN SAL > 4000 THEN 'ABOVE 4000'
          END AS SALRAGE ,COUNT(*) AS CNT
FROM EMP
GROUP BY CASE
        WHEN SAL BETWEEN 0 AND 2000 THEN '0-2000'
           WHEN SAL BETWEEN 2001 AND 4000 THEN '2001-4000'
           WHEN SAL > 4000 THEN 'ABOVE 4000'
           END
```

0-2000 8

2001-4000 5

ABOVE 4000 1

Assignment :-

PERSONS

AADHARNO NAME GENDER AGE ADDR CITY STATE

1 display state wise population ?

2 display gender wise population ?

3 display state wise and with in state gender wise population and also display state and gender wise subtotals ?

4 display age group wise population ?

0-20 ?

21-40 ?

41-60 ?

SALES

DATEID PRODID CUSTID QTY AMOUNT

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

13-sep-21

Integrity Constraints

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

=> used to prevent users from entering invalid data

=> used to enforce rules like min sal must be 3000

Integrity Constraints :-

NOT NULL

UNIQUE

PRIMARY KEY

CHECK

FOREIGN KEY

DEFAULT

=> above constraints can be declared in two ways

1 column level

2 table level

column level :-

=> if constraint is declared immediately after declaring column then it is called column level.

```
CREATE TABLE <tabname>
 (
   COLNAME DATATYPE(SIZE) CONSTRAINT,
   COLNAME DATATYPE(SIZE) CONSTRAINT,
   ----------
  );
```

NOT NULL :-

=> NOT NULL constraint doesn't accept nulls.

=> a column declared with NOT NULL is called mandatory column

```
ex :- CREATE TABLE emp11
      (
         empid  int,
         ename  varchar(10) NOT NULL
      );
```

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

UNIQUE :-

=> UNIQUE constraint doesn't accept duplicates

=> a column declared with UNIQUE into that column duplicates are not allowed

```
ex :- CREATE TABLE emp12
     (
          empid  int,
          emailid  varchar(30)  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
```

NOTE :- UNIQUE constraint allows one NULL.

PRIMARY KEY :-

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

=> PRIMARY KEY is the combination of unique & not null

PRIMARY KEY = UNIQUE + NOT NULL

```
CREATE TABLE emp13
(
  empid  int PRIMARY KEY,
  ename  varchar(10),
  sal    money
)

INSERT INTO emp13 VALUES(100,'A',5000)
INSERT INTO emp13 VALUES(101,'B',1000)
INSERT INTO emp13 VALUES(100,'C',1000) => ERROR
INSERT INTO emp13 VALUES(NULL,'D',1000) => ERROR
```

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

```
CREATE TABLE customers
(
  accno         int  PRIMARY KEY,
```

```
    name          varchar(10) NOT NULL,
    aadharno      bigint UNIQUE NOT NULL,
    panno       CHAR(10) UNIQUE NOT NULL
  )
```

14-sep-21

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

CHECK constraint :-

=> use check constraint for rules based on conditions.

syn :- CHECK(condition)

Example 1 :- sal must be min 3000

```
CREATE TABLE emp15
(
  empno int ,
  ename varchar(10),
  sal   money CHECK(sal>=3000)
)
```

```
INSERT INTO emp15 VALUES(100,'A',5000)
INSERT INTO emp15 VALUES(101,'B',1000) => ERROR
INSERT INTO emp15 VALUES(102,'C',NULL) => ACCEPTED
```

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

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

Example 3 :- pwd must be min 8 chars ?

```
      pwd  varchar(12) CHECK(LEN(pwd)>=8)
```

Example 4 :- emailid must contain '@'
must end with '.com' or '.co' or '.in' ?

```
          emailid varchar(30) check(emailid like '%@%'
                                    and
                                    (
                                      emailid like '%.com'
                                      or
                                      emailid like '%.co'
```

```
                    or
                    emailid like '%.in'
                   ))
```

FOREIGN KEY :-

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

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

example :-

PROJECTS

projid	pname	duration	client	cost
100	ABC	5 YEARS	TATA MOTORS	300
101				
102				

EMP

empid	ename	sal	projid REFERENCES projects(projid)
1	A	5000	100
2	B	6000	101
3	C	7000	999 => not accepted
4	D	3000	100 => accepted
5	E	4000	NULL => accepted

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

=> foreign key allows duplicates and nulls

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

=> primary key table is parent and fk table is child.

Example :-

```
CREATE TABLE projects
(
 projid  int  PRIMARY KEY,
 name    varchar(20),
 duration varchar(20)
)

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

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

INSERT INTO emp_proj VALUES(1,'A',5000,100);
INSERT INTO emp_proj VALUES(2,'B',4000,999); => ERROR
```

INSERT INTO emp_proj VALUES(3,'C',5000,100); => ACCEPTED

INSERT INTO emp_proj VALUES(4,'D',5000,NULL); => ACCEPTED

15-SEP-21

establishing one to one relationship :-

=> by default sql server creates one to many relationship , to establish one to one relationship declare foreign key with unique constraint.

Example :-

DEPT

DNO	DNAME
10	HR
20	IT
30	SALES

MGR

MGRNO	MNAME	DNO REFERENCES DEPT(DNO) UNIQUE
1	A	10
2	B	20
3	C	30

DEFAULT :-

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

hiredate date default getdate()

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

```
CREATE TABLE emp22
(
  empno int,
  hiredate date default getdate()
)

insert into emp22 (empno) values(100)
insert into emp22 values(101,null)
insert into emp22 values(102,'2021-01-01')

select * from emp22
```

100 2021-09-15
101 null
102 2021-01-01

Assignment :-

ACCOUNTS
ACCNO ACTYPE NAMEBAL

rules :-

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

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 default tdate must be today's date

4 tamt must be multiple of 100

5 accno should match with accounts table accno

6 accno should not be null

TABLE LEVEL :-

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

=> use table level to declare constraint for multiple or combination of columns.

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

Declaring check constraint at table level :-

MANAGERS

MGRNO	MNAME	START_DATE	END_DATE
100	A	2021-09-15	2021-01-01 => INVALID

RULE :- end_date > start_date

=> in the above example constraint is based on multiple column so cannot be declared at column and must be declared at table level.

```
CREATE TABLE managers
(
  mgrno int primary key,
  mgrname varchar(10),
  start_date date ,
  end_date date ,
        CHECK(end_date > start_date)
)
```

composite primary key :-

=> sometimes in tables we can't uniquely identify by using single column and we need combination of columns to uniquely identify , so that combination should be declared primary key.

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

=> composite primary key declared at table level.

Example :-

ORDERS

ordid	ord_dt	del_dt	cid
1000	10-	20-	10
1001	12-	22-	11
1002	15-	25-	12

PRODUCTS

prodid	pname	price
100	A	2000
101	B	3000
102	C	5000

ORDER_DETAILS

ordid	prodid	qty
1000	100	1
1000	101	1
1000	102	1
1001	100	1
1001	101	1

=> in the above ordid,prodid combination uniquely identifies the records so declare this combination

as primary key at table level.

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

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

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

INSERT INTO products VALUES(100,'A',1000)
INSERT INTO products VALUES(101,'B',2000)

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

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

A UNIQUE

B CHECK

C NOT NULL

D PRIMARY KEY

E FOREIGN KEY

ANS :- C

Adding constraints to existing table :-

"ALTER" command is used to add constraint to existing table.

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

Adding primary key :-

=> primary key cannot be added to nullable column

=> to add pk first change the column to not null then add pk

STEP 1 :-

```
  ALTER TABLE emp88
     ALTER COLUMN empno INT NOT NULL
```

STEP 2:-

```
  ALTER TABLE emp88
```

ADD PRIMARY KEY(empno)

Adding check constraint :-

=> add check constraing with condition sal>=3000

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

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

=> above command fails because some of the values are less than 3000 , while adding constraint

sql server also validates existing data , if existing data satisifies condition then constraint

is added otherwise not.

WITH NOCHECK :-

=> if check constraint is added with NO CHECK 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 foreign key to column dno that refers dept table primary key ?

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

changing from NULL to NOT NULL :-

=> modify the column ename from NULL to NOT NULL ?

```
ALTER TABLE emp88
 ALTER COLUMN ename VARCHAR(10) NOT NULL
```

17-sep-21

Droping constraints :-

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

=>drop check constraint in emp88 table ?

```
ALTER TABLE emp88
  DROP CONSTRAINT CK__emp88__sal__440B1D61
```

=> drop primary key in emp88 table ?

```
alter table emp88
   drop constraint PK__emp88__AF4C318A73E3B11F
```

=> drop primary key in dept table ?

alter table dept

drop PK__DEPT__E0EB08D7D06DCA8C => ERROR

drop table dept => ERROR

truncate table dept => ERROR

NOTE :- primary key constraint cannot be dropped if referenced by some fk

primary key table cannot be dropped if referenced by some fk

primary key table cannot be truncated if referenced by some fk

DELETE RULES :-

ON DELETE NO ACTION (DEFAULT)

ON DELETE CASCADE

ON DELETE SET NULL

ON DELETE SET DEFAULT

=> delete rules are declared with foreign key constraint

=> delete rule specifies how child rows are affected if we delete parent row

ON DELETE NO ACTION :-

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

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

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

CREATE TABLE emp99
(
  empno int primary key,
  ename varchar(10),
  dno int references dept99(dno)
)

INSERT INTO emp99 VALUES(1,'A',10),(2,'B',10)
```

DELETE FROM DEPT99 WHERE DNO=10 => ERROR

scenario :-

ACCOUNTS

ACCNO	BAL
100	10000
101	20000

LOANS

ID	TYPE	AMT	ACCNO

1 H 30 100

2 C 10 100

ON DELETE CASCADE :-

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

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

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

CREATE TABLE emp99
(
  empno int primary key,
  ename varchar(10),
  dno int references dept99(dno)
        ON DELETE CASCADE
)

INSERT INTO emp99 VALUES(1,'A',10),(2,'B',10)

DELETE FROM DEPT99 WHERE DNO=10  => 1 row affected

SELECT * FROM emp99  => no rows selected
```

scenario :-

ACCOUNTS

ACCNO BAL

100 10000

101 20000

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

1 W ?/ 1000 100

2 D ?? 2000 100

ON DELETE SET NULL :-

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

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

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

CREATE TABLE emp99
(
```

```
  empno int primary key,
  ename varchar(10),
  dno int references dept99(dno)
        ON DELETE SET NULL
)

INSERT INTO emp99 VALUES(1,'A',10),(2,'B',10)
```

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

SELECT *FROM emp99

1	A	NULL
2	B	NULL

scenario :-

PROJECTS

projid	pname	duration
100		
101		
102		

EMP

empid	ename	projid
1		100
2		101

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

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

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

CREATE TABLE emp99
(
  empno int primary key,
  ename varchar(10),
  dno int default 20
        references dept99(dno)
        ON DELETE SET DEFAULT
)

INSERT INTO emp99 VALUES(1,'A',10),(2,'B',10)

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

SELECT * FROM EMP99
```

EMPNO	ENAME	DNO
1	A	20

2 B 20

UPDATE rules :-

ON UPDATE NO ACTION

ON UPDATE CASCADE

ON UPDATE SET NULL

ON UPDATE SET DEFAULT

=> specifies how foreign key value is affected if primary key updated

---

18-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.

Types of joins :-

---

1 inner join or equi join

2 outer join

left join

right join

full join

3 non equi join

4 self join

5 cross join or cartesian join

20-sep-21

Equi Join / Inner Join :-

=> to perform 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 also not compulsory.

=> equi join is performed between the two tables based on the common column with same datatype.

syntax :-

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

join condition :-

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

=> join condition determines which record of 1st table should be joined with 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	6000	20
3	C	4000	30
4	D	3000	10
5	E	4000	NULL

DEPT

DEPTNO	DNAME
10	ACCOUNTS
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 6000 RESEARCH ??

3 C 4000 SALES ??

4 D 3000 ACCOUNTS ??

=> display EMPNO ENAME SAL DEPTNO DNAME LOC ?

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

=> 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 two tables :-

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

conditions required.

```
SELECT columns
FROM tab1,tabl2,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 ?

```
          ----- -----    -------------     --------
        EMP       DEPT     LOCATIONS     COUNTRIES
```

```
SELECT e.ename,
      d.dname,
      l.city,l.state,
      c.country_name as country
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
```

21-sep-21

join styles :-

---

1 Native style (SQL SERVER style)

2 ANSI style

ANSI style :-

=> Adv of ANSI style is portability.

=> Native style doesn't gurantees portability but ANSI style gurantees portability

=> 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

use WHERE clause for filter conditions

display ENAME DNAME CITY STATE COUNTRY ?

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

OUTER JOIN :-

=> inner join returns only matching records but won't return unmatched records , to get unmatched

records also perform outer join.

=> outer join is possible in ANSI style

EMP

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

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTS	
20	RESEARCH	
30	SALES	
40	OPERATIONS => unmateched row	

=> outer join 3 types

1 left join

2 right join

3 full join

left join :-

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

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

=> returns all rows 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 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
```

=> returns all rows from emp & dept

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

22-sep-21

display unmatched records from emp table ?

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

display unmatched records from dept table ?

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

display unmatched records from both tables ?

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

```
WHERE d.dname IS NULL
      OR
      e.ename 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 display projects where no 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 not based on "=" operator and it 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 EMPNO ENAME SAL GRADE ?

---------- -----

EMP SALGRADE

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

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

=> display grade 3 employee details ?

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

=> display ENAME DNAME   GRADE ?

------ ----- ------

EMP DEPT SALGRADE

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

23-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.

Example :-

EMP

EMPNO	ENAME	MGR
7369	SMITH	7902
7499	ALLEN	7698

7521	WARD	7698
7566	JONES	7839
7654	MARTIN	7698
7698	BLAKE	7839
7782	CLARK	7839
7788	SCOTT	7566
7839	KING	NULL
7902	FORD	7566

=> above table contains manager numbers but to display manager names we need to perform self join

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

FROM EMP X,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
JONES KING
MARTIN BLAKE

=> display employees reporting to blake ?

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

=> display blake's manager name ?

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

Assignment :-

=> display employees earning more than their managers ?

=> display employees joined before their manager ?

=> display manager name wise no of employees reporting ?

=> display manager name where no of employees reporting is more than 4 ?

=>

TEAMS

ID	COUNTRY
1	IND
2	AUS
3	RSA

write a query to display following output ?

IND VS AUS

IND VS RSA

AUS VS RSA

CROSS 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 is performed between two tables then each record of table1 joined with each and

every record of table2.

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

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

GROUP BY & JOIN :-

=> display dept wise no of employees in the 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
```

24-sep-21

subqueries / nested queries :-

=> writing a query in another query is called subquery or nested query

=> one query is called outer/parent/main query

=> other query is called inner/child/sub query

=> sql server first executes inner query then sql server 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 subqueries

single row subqueries :-

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

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

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

=> display employees earning more than blake ?

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

=> display employees who are senior to king ?

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

=> display employee name earning max salary ?

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

=> display employee name having max experience ?

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

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

=> delete the employee having max experience ?

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

=> 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 (7521,7499)
```

27-sep-21

Multi-row subqueries :-

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

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

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

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

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

ANY operator :-

=> use ANY operator when comparision with any value i.e. atleast one

WHERE X > ANY (1000,2000,3000)

IF X=800 FALSE
1500 TRUE
4500 TRUE

WHERE X < ANY(1000,2000,3000)

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

ALL operator :-

=> use ALL operartor when comparision with all value

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

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 return by outer query

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

1 returns a row from outer query

2 pass value to inner query

3 executes inner query

4 returns value to outer query

5 execute 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 empno,ename,sal,deptno
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 empno,ename,sal,deptno
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

28 -sep-21

=> 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 EMP B

SAL SAL

5000 5000 3 > (0) TRUE

1000 1000 3 > (4) FALSE

3000	3000	3 > (2)	TRUE
2000	2000	3 > (3)	FALSE
4000	4000	3 > (1)	TRUE

=> display 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 the result of one process in another process

3 to join query output with a table

Examples :-

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 returns ranks of all the employee 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
```

column alias cannot be used in where clause 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 5th max salary employee details ?

```
SELECT *
```

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

2 display top 3 employee in each dept based on sal ?

3 display top 3 employee in each dept based on experience ?

4 display first 5 rows from emp table ?

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

display 5th row ?

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

display 5th record to 10th record ?

```
SELECT *
FROM ( SELECT empno,ename,sal ,
        ROW_NUMBER() OVER (ORDER BY empno ASC) AS rno
    FROM emp ) E
WHERE rno BETWEEN 5 AND 10
```

display even no rows ?

```
SELECT *
FROM ( SELECT empno,ename,sal ,
        ROW_NUMBER() OVER (ORDER BY empno ASC) AS rno
    FROM emp ) E
WHERE rno%2=0
```

display last record ?

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

29-sep-21

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 derived tables outer query cannot be DML and it must be always SELECT , to overcome this

problem use CTEs

CTE :-

=> A Common Table Expression, also called as CTE in short form, is a temporary named result

set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

=> CTEs are used to simplify the complex operations

=> in derived tables outer query cannot be dml but in CTEs outer query can be DML.

Syn :-

WITH <NAME>

AS

(SELECT STATEMENT),

SELECT/INSERT/UPDATE/DELETE

Example 1 :- delete first 5 rows from emp ?

WITH E

AS

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

Example 2 :- delete duplicate records ?

METHOD 1 :-

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 :- group the rows whose eno,ename,sal are same then with in group generate row numbers

```
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
```

METHOD 2 :-

1 create temp table and copy distinct records to temp table

```
SELECT DISTINCT * INTO TEMP FROM emp22
```

2 truncate original table

```
TRUNCATE TABLE emp22
```

3 copy records from temp to emp22

```
INSERT INTO emp22
SELECT * FROM temp
```

scalar subqueries :-

=> subqueries in SELECT clause are called scalar subqueries

```
SELECT (subquery1),(subquery2),----
```

FROM tabname

WHERE condition

=> subquery output acts like a column for outer query

Example 1 :-

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

emp	dept
14	4

Example 2 :-

display dept wise total sal ?

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

10	8750
20	10875
30	9400

display deptno,dept_totsal,totsal ?

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

10	8750	29025
20	10875	29025
30	9400	29025

Assignments :-

1

T1	T2
F1	C1
1	A
2	B
3	C

=> join the above two tables and display following output ?

1	A
2	B
3	C

2

T1
AMT
1000
-500
2000
-1000
3000

-5000

=> write a query to display following output ?

POS	NEGATIVE
1000	-500
2000	-1000
3000	-5000

30-SEP-21

PIVOT operator :-

=> used to convert rows into columns

=> used for cross tabulation or matrix report

Syntax :-

```
SELECT *
FROM (SELECT STATEMENT) <ALIAS>
PIVOT
 (
  AGGR-EXPR FOR COLNAME IN (V1,V2,V3,--)
 ) AS <PIVOT_TABLE_NAME>
ORDER BY COL ASC/DESC
```

Example 1 :-

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

```
SELECT *
FROM (SELECT deptno,job,sal FROM emp) E
PIVOT
(
  SUM(sal) FOR deptno IN ([10],[20],[30])
) AS PIVOT_TABLE
ORDER BY job ASC
```

Example 2 :-

	1	2	3	4
1980	?	?	?	?
1981				
1982				

1983

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

Example 3 :- converting rows into columns

STUDENT

SNO	SNAME	SUBJECT	MARKS
1	A	MAT	80
1	A	PHY	90
1	A	CHE	70
2	B	MAT	60
2	B	PHY	80
2	B	CHE	70

OUTPUT :-

SNO	SNAME	MAT	PHY	CHE
1	A	80	90	70
2	B	60	80	70

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

01-oct-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 transaction must gurantee a property called atomocity i.e. all or none , if transaction contains multiple dmls if all are successful then it must be saved , if one of the dml fails then entire transaction must be cancelled.

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

1 COMMIT => to save transaction
2 ROLLBACK => to cancel transaction
3 SAVE TRANSACTION => to cancel part of the transaction

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

=> in sql server a txn begins implicitly and ends implilcitly with commit

=> user can also start transaction explicitly by using "BEGIN TRANSACTION" command and end explicitly with COMMIT/ROLLBACK.

Example 1 :-

```
CREATE TABLE a(a int)
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)
```

INSERT INTO a VALUES(60)

ROLLBACK

output :- all operations are implicitly committed

Example 2 :-

CREATE TABLE a(a int) => implicitly committed

BEGIN TRANSACTION => txn begins 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)

INSERT INTO a VALUES(60)

ROLLBACK => txn ends

=> if txn ends with rollback then all the changes made in transaction are cancelled

Example 3 :-

CREATE TABLE a(a int) => implicitly committed

BEGIN TRANSACTION => txn begins t1

INSERT INTO a VALUES(10)

INSERT INTO a VALUES(20)

INSERT INTO a VALUES(30)

COMMIT => txn ends

INSERT INTO a VALUES(40) => implicitly committed

INSERT INTO a VALUES(50) => implicitly committed

INSERT INTO a VALUES(60) => implicitly committed

ROLLBACK

SAVE TRANSACTION :-

=> we can declare save transaction and we can cancel upto the save transaction

=> using this 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
```

02-oct-21

DB security :-

---

1 logins => provides security at server level

2 users => provides security at db level

3 privileges => provides security at table level

4 views => provides security at row & col level

DB OBJECTS /SCHEMA OBJECTS :-

TABLES

VIEWS

SEQUENCES

INDEXES

VIEWS :-

=> a view is a subset of a table

=> a view is a virtual table because it doesn't store and doesn't occupy memory

=> a view is a representation of a query

=> views are created

1 to provide security

2 to reduce complexity

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

=> views are 2 types

1 simple views

2 complex views

simple views :-

=> a view said to be simple view if based on single table.

syntax :-

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

example :-

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

=> sql server creates view "V1" and stores query but not query output

```
SELECT * FROM V1
```

=> when above query is submitted to sql server ,it rewrite the query as follows

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

Granting permissions on view to user :-

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

VIJAY :-

```
1 SELECT * FROM V1
```

2 INSERT INTO V1 VALUES(9999,'ABC','CLERK',20) => 1 ROW AFFECTED

3 UPDATE V1 SET JOB='MANAGER' WHERE EMPNO=9999

4 UPDATE V1 SET SAL=5000 WHERE EMPNO=9999 => ERROR

ROW LEVEL SECURITY :-

CREATE VIEW V2

AS

SELECT empno,ename,job,deptno

FROM emp

WHERE deptno=20

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

VIJAY :-

INSERT INTO V2 VALUES(8888,'ABC','CLERK',30) => 1 ROW AFFECTED

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

WITH CHECK OPTION :-

=> if view created with "WITH CHECK OPTION" then any dml command through view violates where condition

that dml is not accepted

CREATE VIEW V3

AS

SELECT empno,ename,job,deptno

FROM emp

WHERE deptno=20

WITH CHECK OPTION

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

VIJAY :-

INSERT INTO V3 VALUES(444,'XYZ','CLERK',30) => ERROR

complex views :-

=> a view said to be complex view

1 if based on multiple tables

2 if query contains group by clause

having claue

distinct clause

aggregate functions

subqueries

=> view reduces complexity , with the help of views complex queries can be converted into simple queries

Example 1 :-

CREATE VIEW CV1

AS

SELECT e.empno,e.ename,e.sal,

d.deptno,d.dname,d.loc

```
FROM emp e INNER JOIN dept d
  ON e.deptno = d.deptno
```

=> after creating view whenever we want 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
```

=> after creating view , whenever we want dept wise summary 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 not updatable

=> display list of views created by user ?

SELECT * FROM INFORMATION_SCHEMA.VIEWS

Droping views :-

DROP VIEW V1

=> if base table is dropped what about views created on base table ?

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

WITH SCHEMABINDING :-

=> if view created with schemabinding then sql server will not allow user to drop table if any view

exists on the table.

Rules :-

1 "*" is not allowed in select

2 tablename should be prefixed with schemaname

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

DROP TABLE dept => ERROR

SEQUENCE :-

=> a sequence is also a db object created to generate sequence numbers

=> used to auto increment column values

syn :-

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

Ex :-

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

MAXVALUE 5

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

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

```
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 the above sequence to update employee numbers ?

```
UPDATE emp SET empno = NEXT VALUE FOR S2
```

calling sequence in expressions :-

INVOICE

INVNO INV_DT

MLN/1005/01 ?

MLN/1005/02 ?

```
CREATE SEQUENCE S3
START WITH 100
INCREMENT BY 1
MAXVALUE 1000

CREATE TABLE invoice
(
 invno  varchar(20).
 inv_dt datetime
)

INSERT INTO INVOICE
    VALUES('MLN/' + format(getdate(),'MMyy') + '/' + CAST(NEXT VALUE FOR S3 AS VARCHAR),GETDATE())
```

How to reset sequence ?

1 manually

2 automatically

manual :-

ALTER SEQUENCE S3 RESTART WITH 100

Automatic using CYCLE option :-

=> by default sequence created with NOCYCLE , it starts from start with and generates upto to max

after reaching max then it stops.

=> if sequence created with cycle then it starts from start with and generates upto max and afer

reaching max then it will be reset to min.

CREATE SEQUENCE S4

START WITH 100

INCREMENT BY 1

MAXVALUE 1000

MINVALUE 1

CYCLE

08-oct-21

INDEXES :-

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

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

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

=> indexes are created on columns

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

=> when we submit a query to sql server , it goes through following phases

1 parsing
2 optimization
3 execution

parsing :-

---

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

optimization :-

=> sql server prepares plans to execute the query

1 table scan

2 index scan

=> estimates the cost of each plan and selects best plan i.e. plan that takes less cost

execution :-

=> executor executes the query according to plan selected by optimizer

Types of Indexes :-

---

1 Non Clustered

simple

composite

unqiue

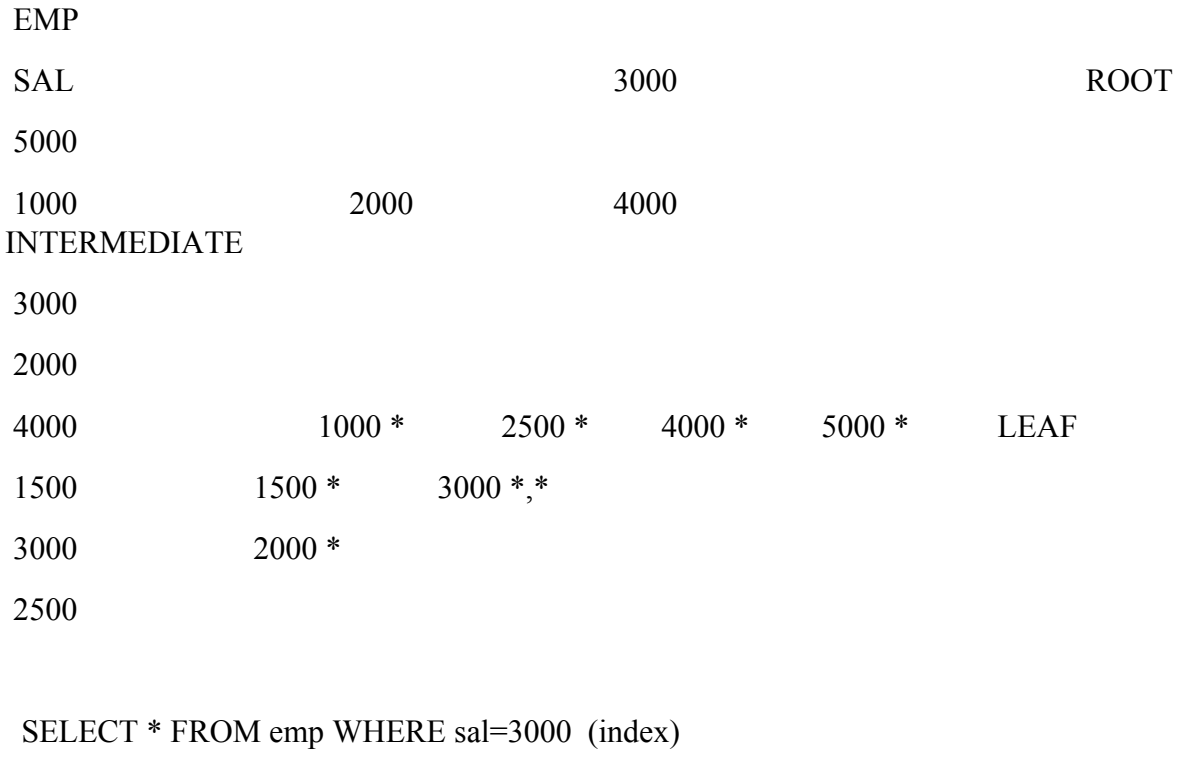
2 Clustered

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)



SELECT * FROM emp WHERE sal=3000 (index)

SELECT * FROM emp WHERE sal>=3000 (index)

SELECT * FROM emp WHERE sal<=3000 (index)

composite index :-

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

ex :- 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

ex :- CREATE UNIQUE INDEX I3 ON EMP(ENAME)

K

G Q

ADAMS * JAMES * MARTIN * SCOTT *

ALLEN * JONES * MILLER * SMITH *

BLAKE * KING *

SELECT * FROM EMP WHERE ENAME='BLAKE' ;

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

=> what are the different methods to enforce uniqueness ?

1 declare primary key/unique constraint

2 create unique index

=> primary key/unique columns are implicitly indexed by sql server and sql server creates a unique

index on primary key/unique columns and unique index doesn't allow duplicate so primary key/unique

also doesn't allow duplicates.

CLUSTERED INDEX :-

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

actual records.

=> in non clustered index order of the records in table and order of the records in index will not be

same but where as in clustered in this order will be same.

```
Ex :-   CREATE TABLE cust
        (
          cid int,
          cname varchar(10)
        )

        CREATE CLUSTERED  INDEX I4 ON CUST(CID)

        INSERT INTO CUST VALUES(10,'A')
           INSERT INTO CUST VALUES(80,'B')
           INSERT INTO CUST VALUES(20,'C')
           INSERT INTO CUST VALUES(60,'D')
        INSERT INTO CUST VALUES(50,'E')
```

30 70

10 A 50 E 60 D 80 B

20 C

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

=> by default sql server creates clustered index on primary key

=> sql server allows one clustered index per table

diff between clustered and non clustered indexes ?

	non clustered	clustered
1	stores pointers to actual records	stores actual reacords
2	order of the records in index and table will not be same	order will be same
3	needs extra storage	doesn't extra storage
4	requires two lookups	requires only one lookup
5	slower compre to clustered index	faster

6	999 non clustered indexes allowed per table	only one clustered index is allowed per table
7	created explicitly	created implicitly on primary key column

=> display list of indexes created on emp table ?

sp_helpindex emp

Droping index :-

DROP INDEX EMP.I1

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

ans :- indexes are also dropped

11-oct-21 TSQL programming

Features :-

---

1 improves performance :-

=> in tsql , sql commands can be grouped into one program and we submit that program to sql server

so in tsql no of requests and response between user and sql server are reduced and performance

is improved.

2 supports conditional statements :-

=> tsql supports conditional statements like if-then-else

3 supports loops :-

=> loops are used to execute statements repeatedly multiple times. TSQL supports loops like while loop

4 supports error handling :-

=> in tsql programs if any statement causes runtime error then we can handle that error and we can

replace system generated 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 these programs are stored in db so only authorized users can execute these programs.

=> TSQL blocks are 2 types

1 anonymous blocks

2 named blocks

stored procedures

stored functions

triggers

Anonymous blocks :-

=> a tsql block without name is called anonymous block

=> the following statements are used in TSQL programming

1 DECLARE

2 SET

3 PRINT

DECLARE statement :-

=> statement used to declare variables

syn :- DECLARE @variablename datatype(size)

Ex :- DECLARE @a int

```
DECLARE @s varchar(10)
DECLARE @d date

DECLARE @a int,@s varchar(10),@d date
```

SET statement :-

=> statement used to assign value to variable

```
SET @var = value
```

ex :-
```
SET @a = 1000
SET @s = 'abc'
SET @d = GETDATE()
```

PRINT statement :-

=> used to print messages or variable values

```
PRINT @a
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,merge)

2 DRL (select)

3 TCL (commit,rollback,save transaction)

SELECT stmt syntax :-

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

ex :-

1

```
SELECT @s=sal
FROM emp
WHERE empno=7369
```

2

```
SELECT @n=ename,@s=sal
FROM emp
WHERE empno=7369
```

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

```
DECLARE @eno int,@name varchar(10),@sal money
SET @eno=110
SELECT @name=ename,@sal=sal FROM emp WHERE empno=@eno
PRINT @name + '   ' + CAST(@sal AS VARCHAR)
```

18-oct-21

=> write a prog to input empno and calculate experience ?

```
DECLARE @eno int,@hire date,@expr tinyint
SET @eno=110
SELECT @hire=hiredate FROM emp WHERE empno=@eno
SET @expr = DATEDIFF(yy,@hire,GETDATE())
PRINT CAST(@expr as varchar) + ' years '
```

conditional statements :-

---

1 IF-ELSE

2 MULTI IF

3 NESTED IF

IF-ELSE :-

```
IF COND
 BEGIN
  STATEMENTS
 END
ELSE
 BEGIN
   STATEMENTS
 END
```

MULTI-IF :-

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

```
 BEGIN
   STATEMENTS
 END
ELSE
 BEGIN
   STATEMENTS
 END
```

NESTED IF :-

```
IF COND
 BEGIN
    IF COND
     BEGIN
       STATEMENTS
      END
    ELSE
      BEGIN
       STATEMENTS
       END
  END
ELSE
  BEGIN
    STATEMENTS
  END
```

=> write a prog to input empno and increment employee sal by specific amount and after increment if

sal exceeds 5000 then cancel that increment ?

```
DECLARE @eno int,@amt money,@sal money
SET @eno=107
SET @amt=1000
BEGIN TRANSACTION
UPDATE emp SET sal=sal+@amt WHERE empno=@eno
SELECT @sal=sal FROM emp WHERE empno=@eno
IF @sal>5000
  ROLLBACK
ELSE
  COMMIT
```

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

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

```
DECLARE @eno int,@job varchar(10),@pct int
SET @eno=101
SELECT @job=job FROM emp WHERE empno=@eno
IF @job='CLERK'
  SET @pct=10
ELSE IF @job='SALESMAN'
  SET @pct=15
ELSE IF @job='MANAGER'
  SET @pct=20
ELSE
  SET @pct=5
UPDATE emp SET sal=sal+(sal*@pct/100) WHERE empno=@eno
```

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

ACCOUNTS

ACCNO BAL

100 10000

101 20000

```
DECLARE @acno int,@type char(1),@amt money,@bal money
SET @acno=100
SET @type='w'
SET @amt=1000
IF @type='w'
BEGIN
  SELECT @bal=bal FROM accounts WHERE accno=@acno
  IF @amt > @bal
    PRINT 'insufficient balance'
  ELSE
    UPDATE accounts SET bal=bal-@amt WHERE accno=@acno
END
ELSE IF @type='d'
    UPDATE accounts SET bal=bal+@amt WHERE accno=@acno
ELSE
    PRINT 'invalid transaction'
```

19-oct-21

=> write a prog for 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=@tacno
  SET @cnt2 = @@ROWCOUNT
  IF @cnt1=1 and @cnt2=1
   COMMIT
  ELSE
   ROLLBACK
 END
```

Assignment :-

STUDENT

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

RESULT

SNO	TOTAL	AVG	RESULT

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

WHILE LOOP :-

=> loops are used to execute statemnets repeatedly multiple times

```
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 ?

```
DECLARE @d1 date,@d2 date
SET @d1='2022-01-01'
SET @d2='2022-12-31'
```

```
WHILE(@d1<=@d2)
BEGIN
   PRINT @d1 + '  ' + DATENAME(dw,@d1)
   SET @d1 = @d1 + 1
END
```

Assignment :-

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

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

input :- NARESH

output :-

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

```
DECLARE @s VARCHAR(20),@x int=1
SET @s='NARESH'
WHILE(@x<=LEN(@s))
BEGIN
```

```
  PRINT SUBSTRING(@s,@x,1)
  SET @x=@x+1
 END
```

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

```
  INPUT :- NARESH
  OUTPUT :-

 N
 NA
 NAR
 NARE
 NARES
 NARESH


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

Assignment :-

=> write a prog to print following output ?

```
 1
```

22

333

4444

55555

20-oct-21

CURSORS :-

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

=> from tsql program if we submit a query to sql server , it goes to db and fetch the data and copy

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

DECLARE CURSOR :-

---

syn :- DECLARE <NAME> CURSOR FOR SELECT STATEMENT

Ex :- DECLARE C1 CURSOR FOR SELECT * FROM emp

OPEN CURSOR :-

---

syn :- OPEN <CURSOR-NAME>

Ex :- OPEN C1

1 select statement 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 temporary memory

FETCHING RECORDS FROM CURSOR :-

=> "FETCH" statement is used to fetch record 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 multiple rows fetch statement should

be executed multiple times , so fetch statement should be inside a loop.

CLOSE CURSOR :-

---

syn :- CLOSE <CURSOR-NAME>

Ex :- CLOSE C1

DEALLOCATE CURSOR :-

---

syn :- DEALLOCATE <CURSOR-NAME>

ex :- DEALLOCATE C1

@@FETCH_STATUS :-

=> system variable used to check whether fetch is successful or not

=> returns 0 or -1

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,@t MONEY=0
  OPEN C1
  FETCH NEXT FROM C1 INTO @sal
  WHILE(@@FETCH_STATUS=0)
  BEGIN
    SET @t = @t + @sal
    FETCH NEXT FROM C1 INTO @sal
  END
    PRINT @t
    CLOSE C1
    DEALLOCATE C1
```

Assignment :-

=> write a prog to calculate max salary ?

=> write a prog to calculate min salary ?

21-oct-21

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

STUDENT

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

RESULT

SNO	TOTAL	AVG	RESULT

```
DECLARE C1 CURSOR FOR SELECT sno,s1,s2,s3 FROM student
DECLARE @sno int,@s1 int,@s2 int,@s3 int,@total int,@avg decimal(5,2),@res char(4)
OPEN C1
FETCH NEXT FROM C1 INTO @sno,@s1,@s2,@s3
WHILE(@@FETCH_STATUS=0)
BEGIN
 SET @total = @s1+@s2+@s3
 SET @avg   = @total/3
 IF @s1>=35 AND @s2>=35 AND @s3>=35
  SET @res='pass'
 ELSE
  SET @res='fail'
 INSERT INTO RESULT VALUES(@sno,@total,@avg,@res)
 FETCH NEXT FROM C1 INTO @sno,@s1,@s2,@s3
END
 CLOSE C1
 DEALLOCATE C1
```

Assignment :-

RAISE_SALARY

EMPNO	PCT
7369	15
7499	20
7521	10
7566	20
7654	15

=> write a prog to increment employee salaries based on the pct in raise_salary table ?

SCROLLABLE CURSOR :-

=> by default cursor is FORWARD ONLY cursor and it supports forward navigation but doesn't support

backward navigation.

=> if cursor declared with SCROLL then it is called SCROLLABLE cursor and it supports both forward

backward navigation.

=>forward only cursor supports only FETCH NEXT statement but SCROLLABLE cursor supports the following

fetch statements.

FETCH FIRST => fetches first record

FETCH NEXT => fetches next record

FETCH PRIOR => fetches previous record

FETCH LAST => fetches last record

FETCH ABSOLUTE N => fetches Nth record from first record

FETCH RELATIVE N => fetches Nth record from current record

Example 1 :-

```
DECLARE C1 CURSOR SCROLL FOR SELECT ename FROM emp
DECLARE @name VARCHAR(10)
OPEN C1
FETCH FIRST FROM C1 INTO @name
PRINT @name
FETCH ABSOLUTE 5 FROM C1 INTO @name
PRINT @name
FETCH RELATIVE 5 FROM C1 INTO @name
PRINT @name
FETCH LAST FROM C1 INTO @name
PRINT @name
FETCH PRIOR FROM C1 INTO @name
PRINT @name
CLOSE C1
DEALLOCATE C1
```

Example 2 :-

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

```
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
```

Assignment :-

=> write a prog to print names from last to first ?

```
 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
```

23-oct-21

ERROR HANDLING / EXCEPTION HANDLING :-

---

1 syntax errors

2 logical errors

3 runtime errors

=> errors that are raised during program execution are called runtime errors

ex :- declare @a tinyint

set @a=1000 => runtime error

=> if any statement causes runtime error then sql server display error message and continues program

execution , to replace system generated error message with our own simple and user friendly message

we need to handle runtime error.

=> to handle runtime error include a block called TRY---CATCH block

```
BEGIN TRY
  statement 1
  statement 2
  statement 3    => statements causes exception
  statement 4
END TRY
BEGIN CATCH
  statements     => statements handles exception
END CATCH
```

=> if any statement in try block causes exception 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=20
SET @c=@a/@b
PRINT @c
END TRY
BEGIN CATCH
 PRINT 'ERROR'
END CATCH
```

ERROR HANDLING FUNCTIONS :-

---

1 ERROR_NUMBER() => returns error number

2 ERROR_MESSAGE() => returns error message

3 ERROR_SEVERITY() => returns error severity level

4 ERROR_STATE() => returns error state

5 ERROR_LINE() => returns line number

Example 2 :-

```
DECLARE @a tinyint,@b tinyint,@c tinyint
BEGIN TRY
SET @a=100
SET @b=20
SET @c=@a/@b
```

```
PRINT @c
END TRY
BEGIN CATCH
 IF ERROR_NUMBER()=220
    PRINT 'value exceeding limit'
 ELSE IF ERROR_NUMBER()=8134
    PRINT 'divisor cannot be zero'
END CATCH
```

Example 3 :-

```
CREATE TABLE emp99
(
 empno int PRIMARY KEY,
 ename VARCHAR(10) NOT NULL,
 sal   MONEY CHECK(sal>=3000)
)
```

=> write a prog to insert data into emp99 table ?

```
DECLARE @eno int,@name varchar(10),@sal money
BEGIN TRY
SET @eno=101
SET @name='PQR'
SET @sal=6000
INSERT INTO emp99 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 must be min 3000'
END CATCH
```

RAISERROR :-

=> procedure used to raise our own error

=> errors raised by user are called user defined errors

=> user defined errors raised by user by using RAISERROR procedure

syn :- RAISERROR(error msg,severity level,state)

severity – the severity of an error.

0-10 – informational messages

11-18 – errors

19-25 – fatal errors

state – the unique identification number that you can use to identify the code section that is causing the error and values are between 0 and 255.

```
DECLARE @eno int,@name varchar(10),@sal money
DECLARE @msg varchar(100),@errsvr int,@errst int
BEGIN TRY
```

```
 SET @eno=101
 SET @name='PQR'
 SET @sal=6000
 INSERT INTO emp99 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 must be min 3000'
  SET @errsvr = ERROR_SEVERITY()
  SET @errst  = ERROR_STATE()
  RAISERROR(@msg,@errsvr,@errst)
END CATCH
```

25-oct-21

Named TSQL blocks :-

---

1 stored procedures

2 stored functions

3 triggers

sub-programs :-

---

1 stored procedures

2 stored functions

Advantages :-

---

1 modular programming :-

=> with the help of procedures & function a big tsql program can be divided into small modules

2 reusability :-

=> proc & func can be centralized i.e. stored in db and applications which are connected to db can

reuse procedures & functions.

3 security :-

=> because these programs are stored in db so they are secured only authorized users can executes

these programs

4 invoked from front-end applications :-

=> proc & func can be invoked from front-end applications like java,.net,php

5 improves performance :-

=> procedures improves performance because they are precompiled i.e. compiled already and ready for

execution. when we create a procedure program is compiled and stored in db and whenever we call

procedure only execution is repeated but not compilation.

stored procedures :-

=> procedure is a named tsql block that accepts some input performs some action and may or may not

returns a value.

=> procedures are created to perform one or more dml operations on db.

syn :-

CREATE OR ALTER PROCEDURE <NAME>

parameters if any

AS

STATEMENTS

parameters :-

=> we can declare parameters and we can pass values to parameters

=> parameters are 2 types

1 INPUT

2 OUTPUT

INPUT :-

=> always recieves value

=> default

=> read only

OUTPUT :-

=> always sends value

=> write only

Example 1 :-

=> create procedure to increment specific employee sal by specific amount ?

```
CREATE OR ALTER PROCEDURE raise_salary
@eno int,
@amt money
AS
 UPDATE emp SET sal=sal+@amt WHERE empno=@eno
```

Execution :-

---

method 1 :- (positional association)

EXECUTE raise_salary 101,1000

method 2 :- (named association)

EXECUTE raise_salary @eno=101,@amt=1000

Example 2 :-

```
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 101,1000,@s output
print @s
```

26-oct-21

Example 3 :-

ACCOUNTS

ACCNO	ACTYPE	BAL
100	S	10000
101	S	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
```

EXECUTION :-

```
DECLARE @B MONEY
EXEC DEBIT 100,1000,@B OUTPUT
PRINT @B
```

Example 4 :-

=> create a

```
CREATE TABLE emp44
(
  empno int primary key,
  ename varchar(10) not null,
  sal  money check(sal>=3000)
)
```

create a procedure to insert data into emp44 table ?

```
CREATE OR ALTER PROCEDURE insert_emp44
@eno int,
@ename varchar(10),
@sal money,
@msg varchar(100) OUTPUT
AS
 BEGIN TRY
  INSERT INTO emp44 VALUES(@eno,@ename,@sal)
  SET @msg='record inserted successfully'
 END TRY
 BEGIN CATCH
  SET @msg=ERROR_MESSAGE()
 END CATCH


DECLARE @s VARCHAR(100)
EXECUTE insert_emp44 2,'C',1000,@s OUTPUT
PRINT @s
```

USER DEFINE FUNCTIONS :-

=> functions created by user are called user define functions.

=> when predefine functions not meeting our requirements then we create our own functions called user define functions

=> functions are created

1 for calculations

2 to fetch value from db

=> functions are 2 types

1 scalar valued (SVF)

2 table valued (TVF)

scalar valued functions :-

=> these functions accepts some input performs some calculation and must return a value

=> return type of the functions is 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 programs

3 front-end applications

executing from sql commands :-

```
SELECT DBO.CALC(10,20,'*') => 200
```

Example 2 :- create function to calculate employee experience ?

```
CREATE OR ALTER FUNCTION getExpr(@eno int) RETURNS int
AS
BEGIN
  DECLARE @hire DATE,@expr  int
  SELECT @hire=hiredate FROM emp WHERE empno=@eno
  SET @expr = DATEDIFF(yy,@hire,GETDATE())
  RETURN @expr
END

SELECT DBO.GETEXPR(110)  => 38
```

27-oct-21

=> create a function to calculate order amount of particular order ?

input :- ordid = 1000

output :- order amount = 8000

PRODUCTS

prodid	pname	price
100	A	2000
101	B	1000
102	C	1500

ORDERS

ordid	prodid	qty
1000	100	2
1000	101	1

1000 102 2

1001 100 2

1001 101 3

```
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,@qty int,@price money,@amount money=0
  OPEN C1
  FETCH NEXT FROM C1 INTO @pid,@qty,@price
  WHILE(@@FETCH_STATUS=0)
  BEGIN
    SET @amount = @amount + (@qty*@price)
    FETCH NEXT FROM C1 INTO @pid,@qty,@price
  END
   CLOSE C1
   DEALLOCATE C1
   RETURN @amount
END
```

C1

prodid qty price

100 2 2000

101 1 1000

102 2 1500

SELECT DBO.GETORDAMT(1000) => 8000

TABLE VALUED FUNCTIONS :-

=> these functions returns records

=> return type of these functions must be TABLE

=> return expression must be select statement

=> table valued functions 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 to return list of employees working for specific dept ?

```
CREATE OR ALTER FUNCTION getEmpList(@d int) RETURNS TABLE
AS
 RETURN (SELECT * FROM emp WHERE deptno = @d)
```

```
SELECT * FROM DBO.getEmpList(20)
```

example 2 :- create a function to return top N employee list based on sal ?

```
CREATE OR ALTER FUNCTION getTopNEmpList(@n int) RETURNS TABLE
AS
 RETURN (SELECT *
```

```
FROM (SELECT empno,ename,sal,
      DENSE_RANK() OVER (ORDER BY sal DESC) as rnk
    FROM emp) E
WHERE rnk<=@n)
```

```
SELECT * FROM DBO.GETTOPNEMPLIST(5)
```

28-oct-21

Assignment :-

CUSTOMERS

CUSTID NAME ADDR DOB PHONE EMAILID

ACCOUNTS

ACCNO ACTYPE BAL CUSTID

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

```
CREATE SEQUENCE S1
START WITH 1
INCREMENT BY 1
MAXVALUE 99999
```

=> create following procedures & functions to implement various 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 (scalar func)

7 statement between two given dates (table func)

8 latest N transactions (table func)

=> diff b/w procedure & function ?

	procedure	function
1	may or may not returns a value	must return a value
2	can return multiple values	always returns one value
3	returns values using out parameter	returns value using return statement
4	can execute dml commands	dml commands not allowed in functions
5	cannot be executes from sql commands	can be executed from sql commands
6	created to perform dmls	created for calculations
7	create procedure to update balance	create function to get balance

=> diff b/w scalar and table valued functions ?

	scalar	table
1	returns one value	returns records
2	return type must be scalar type	return type must be table
3	return expression must be scalar variable	return expression must be select statement
4	invoked in SELECT clause	invoked in FROM clause

TRIGGERS :-

=> a trigger is also named TSQL block like procedure but executed implicitly by sql server

=> sql server executes trigger automatically whenever user submits DML/DDL commands

=> triggers are created

1 to control dml/ddls
2 to enforce complex rules & validations
3 to audit tables
4 to manage replicas
5 to generate values for primary key columns

syntax :-

```
CREATE OR ALTER TRIGGER <NAME>
ON <TABNAME>
AFTER/INSTEAD OF INSERT,UPDATE,DELETE
AS
  STATEMENTS
```

AFTER triggers :-

=> if trigger is after then sql server executes the trigger after executing DML

INSTEAD OF trigger :-

=> if trigger is instead of trigger 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 DATEPART(DW,GETDATE())=7
       AND
       DATEPART(HH,GETDATE()) NOT BETWEEN 10 AND 13
     BEGIN
       ROLLBACK
       RAISERROR('only between 10am and 2pm',15,1)
     END
 ELSE IF DATEPART(DW,GETDATE())=1
     BEGIN
       ROLLBACK
       RAISERROR('sunday not allowed',15,1)
     END
```

UPDATE EMP SET SAL=2000 WHERE EMPNO=105

29-oct-21

Example 3 :- create trigger to not to allow to update empno ?

```
CREATE OR ALTER TRIGGER T3
ON EMP
AFTER UPDATE
AS
  IF UPDATE(empno)
  BEGIN
    ROLLBACK
    RAISERROR('cannot be updated',15,1)
  END
```

Magic tables :-

---

1 INSERTED

2 DELETED

=> these two tables are called magic tables because they can be accessed only with in trigger

=> using these two tables in triggers we can access data affected by dml

=> 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(555,'ABC',5000) => INSERTED

empno ename sal

555 ABC 5000

UPDATE emp SET sal=6000 WHERE empno=555 => INSERTED

empno sal

555 6000

DELETED

empno sal

555 5000

DELETE FROM emp WHERE empno=555 => DELETED

empno ename sal

555 ABC 6000

Example 4 :-

=> create trigger to not to allow 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
```

Example 5 :-

=> create trigger to insert details into emp_resign whenever employee resigns from organization ?

EMP_RESIGN

empnoename 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
 INSERT INTO emp_resign
 SELECT empno,ename,hiredate,GETDATE() FROM DELETED
```

Auditing :-

=> Auditing means monitoring day-to-day activities

=> Auditing means capturing changes made to table

=> triggers are created for auditing

```
CREATE TABLE emp_audit
(
  uname          varchar(10),
  operation      varchar(10),
  optime       datetime,
  new_eno      int,
  new_ename    varchar(10),
  new_sal      money,
  old_eno      int,
  old_ename    varchar(10),
  old_sal      money
 )
```

create trigger to audit emp table ?

```
CREATE OR ALTER TRIGGER T6
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)
```

30-oct-21

INSTEAD OF triggers :-

=> if trigger is instead of then sql server executes the trigger instead of executing dml

=> create trigger to not to more than 4 employees in a dept ?

EMP99

ENO	DNO
1	10
2	10
3	10
4	10
5	10 => not allowed

```
CREATE TABLE emp99
(
 eno int,
 dno int
)
```

```
CREATE OR ALTER TRIGGER T7
ON EMP99
INSTEAD OF INSERT
AS
 DECLARE @ENO INT,@DNO INT,@CNT INT
 SELECT @ENO=ENO,@DNO=DNO FROM INSERTED
 SELECT @CNT=COUNT(*) FROM EMP99 WHERE DNO=@DNO
 IF @CNT=4
   RAISERROR('max 4 employees per dept',15,1)
 ELSE
   INSERT INTO EMP99 VALUES(@ENO,@DNO)
```

```
AFTER :-                        INSTEAD OF :-
---------                       -------------

IF COND                         IF COND
 BEGIN                            RAISERROR
   ROLLBACK                       ELSE
   RAISERROR                        DML
 END
```

```
 INSERT INTO EMP99 VALUES(1,10)
 INSERT INTO EMP99 VALUES(2,10)
 INSERT INTO EMP99 VALUES(3,10)
 INSERT INTO EMP99 VALUES(4,10)
 INSERT INTO EMP99 VALUES(5,10)  => ERROR
```

Dynamic SQL :-

=> SQL commands build at runtime are called dynamic sql commands

ex :- DROP TABLE emp (static sql)

```
DECLARE @TNAME VARCHAR(20)
SET @TNAME='EMP'
DROP TABLE @TNAME (dynamic sql)
```

=> Dynamic SQL is useful when we don't know tablenames and column names until runtime.

=> Dynamic SQL commands are executed by using

1 EXEC
2 SP_EXECUTESQL

using EXEC statement :-

=> dynamic sql that we want to execute should be passed as a string to EXEC

```
EXEC ('dynamic sql command')
```

create procedure to drop table ?

```
CREATE OR ALTER PROCEDURE drop_table
```

```
 @TNAME VARCHAR(30)
 AS
   EXEC ('DROP TABLE ' + @TNAME)
```

01-nov-21

create procedure to drop all tables from db ?

```
CREATE OR ALTER PROCEDURE DROP_ALL_TABLES
AS
DECLARE C1 CURSOR FOR SELECT TABLE_NAME
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_TYPE='BASE TABLE'
DECLARE @TNAME VARCHAR(20),@SQLCMD VARCHAR(100)
OPEN C1
FETCH NEXT FROM C1 INTO @TNAME
WHILE(@@FETCH_STATUS=0)
BEGIN
 SET @SQLCMD = ' DROP TABLE ' + @TNAME
 EXEC (@SQLCMD)
 FETCH NEXT FROM C1 INTO @TNAME
END
 CLOSE C1
 DEALLOCATE C1
```

USING SP_EXECUTESQL :-

=> Write a prog to print no of rows in each and every table ?

EMP ??

DEPT ??

CUST ??

```
DECLARE C1 CURSOR FOR SELECT TABLE_NAME
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_TYPE='BASE TABLE'
DECLARE @TNAME VARCHAR(20),@SQLCMD VARCHAR(100),@CNT INT
OPEN C1
FETCH NEXT FROM C1 INTO @TNAME
WHILE(@@FETCH_STATUS=0)
BEGIN
  SET @SQLCMD =  'SELECT @CNT=COUNT(*) FROM ' + @TNAME
    EXECUTE SP_EXECUTESQL @SQLCMD,N'@CNT INT OUTPUT',@CNT=@CNT OUTPUT
  PRINT @TNAME + '  ' + CAST(@CNT AS VARCHAR)
  FETCH NEXT FROM C1 INTO @TNAME
END
  CLOSE C1
  DEALLOCATE C1
```

02-nov-21

BACKUP & RESTORE :-

=>

```
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(500)
OPEN C1
FETCH NEXT FROM C1 INTO @DBNAME
WHILE(@@FETCH_STATUS=0)
BEGIN
  SET @FNAME = 'C:\DATA\' + @DBNAME +'.BAK'
  BACKUP DATABASE @DBNAME TO DISK = @FNAME
  FETCH NEXT FROM C1 INTO @DBNAME
END
  CLOSE C1
  DEALLOCATE C1
```

MERGE command :-

=> used to merge data into a table

=> used to manage replicas (duplicate copies)

=> using merge command we can apply changes made to one table to another table

=> used in ETL applications

scenario :-

02/11/21

CUSTS

CID	CNAME	CITY
10	A	HYD
11	B	MUM

=> create replica for custs ?

SELECT * INTO CUSTT FROM CUSTS

CUSTT

CID	CNAME	CITY
10	A	HYD
11	B	MUM

03/11/21

CUSTS

CID	CNAME	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
 WHEN NOT MATCHED BY SOURCE THEN
  DELETE
```

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)
 WHEN NOT MATCHED BY SOURCE THEN
  DELETE ;
```

EXISTS & NOT EXISTS operators :-

=> used to check whether record exists in the table

```
 SELECT *
 FROM TABNAME
 WHERE EXISTS (SELECT STATEMENT )
```

=> EXISTS returns true/false

TRUE => if inner query returns at least one row

FALSE => if inner query returns 0 rows

Example :-

PRODUCTS

prodid pname price

100

101

102

SALES

dateid	prodid	custid	qty	amount
2021-11-03	100	10	1	2000
	101	11	1	3000

=> display list of products which are participated in sale ?

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

100

101

```
SELECT *
FROM products p
WHERE propdid IN (SELECT prodid FROM sales)
```

NOTE :- EXISTS gives good performance than IN operator

=> display products which are not participated in sale ?

```
SELECT *
FROM products p
WHERE NOT EXISTS (SELECT * FROM sales WHERE prodid = p.prodid)
```