

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

7/3/2023

# SQL SERVER

NOTES

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

**Maddileti**



**03-JUL-23**

## **SQL SERVER :-**

-----

**=> a database is a organized collection of interrelated data. For example  
a univ db stores data related to students,courses,faculty etc.**

## **Types of Databases :-**

-----

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

**=> organizations uses OLTP DB for storing day-to-day transactions  
and OLAP for analysis.**

**=> OLTP for running business and OLAP for analyzing business.**

**=> day-to-day operations on db includes**

**C create  
R read  
U update  
D delete**

## **DBMS :-**

-----

**=> DBMS stands for Database Management System , It is a software  
used to create and to manage database.**

**=> DBMS is an interface between user and database.**

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

## **Data Models :-**

-----

**=> based on the structure of the data data models are 3 types**

- 1 Hierarchical**
- 2 Network**
- 3 Relational**

## **Relational Model :-**

-----

**=> Relational Model introduced by E.F.CODD.**

=> according to E.F.CODD in relational model data must be organized in tables i.e. rows and columns

=> a dbms that supports relational model is called rdbms

**CUST**

**CID NAME ADDR => COLUMNS/FIELDS/ATTRIBUTES**

**10 A HYD**

**11 B BLR**

**12 C DEL => ROW/RECORD/TUPLE**

**DATABASE = COLLECTION OF TABLES**

**TABLE = COLLECTION OF ROWS & COLS**

**ROW = COLLECTION OF FIELD VALUES**

**COLUMN = COLLECTION OF VALUES ASSIGNED TO ONE FIELD**

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

**ex :- ACCNO,EMPID,AADHARNO,PANNO,VOTERID**

**RDBMS softwares :-**

-----

<b>SQL SERVER</b>	<b>from microsoft</b>
<b>ORACLE</b>	<b>from oracle corp</b>
<b>DB2</b>	<b>from IBM</b>
<b>MYSQL</b>	<b>from oracle corp</b>
<b>POSTGRESQL</b>	<b>from postgresql forum</b>
<b>RDS</b>	<b>from amazon</b>

**ORDBMS :-**

-----

=> **Object Relational Database Management System**

**ORDBMS = RDBMS + OOPS (reusability)**

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

**ORDBMS softwares :-**

-----

**SQL SERVER**  
**ORACLE**  
**POSTGRESQL**

**what is SQL SERVER ?**

**SQL SERVER is basically a rdbms product from microsoft and also supports ordbsms features and used to manage database.**

=> **SQL SERVER is used for DB Development & Administration**

## **Development**

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

## **Administration**

Installation of sql server  
creating database  
creating logins  
backup & restore  
export & import  
performance tuning

### **CLIENT / SERVER Architecture :-**

-----

**1 SERVER**

**2 CLIENT**

=> server is a system where sql server software is installed and running.

=> inside server sql server manages database.

=> a client is a system from where users can

**1 connects to server**

**2 submit requests to server**

**3 receives response from server**

**client tool :-**

**SSMS (SQL SERVER MANAGEMENT STUDIO)**

**How to connect to sql server :-**

-----

=> open ssms and enter following details

<b>SERVER TYPE</b>	<b>:-</b>	<b>DB Engine</b>
<b>SERVER NAME</b>	<b>:-</b>	<b>DESKTOP-G2DM7GI</b>
<b>Authentication</b>	<b>:-</b>	<b>WINDOWS / SQL SERVER</b>
<b>LOGIN</b>	<b>:-</b>	<b>SA (SYSTEM ADMIN)</b>
<b>PASSWORD</b>	<b>:-</b>	<b>123</b>

=> click **CONNECT**

**creating database in server :-**

-----

=> in object explorer select **Databases => New Database**

**Enter Database Name :- BATCH12**

=> click **OK**

=> a new database is created with following two files

- 1 DATA FILE (.MDF)
- 2 LOG FILE (.LDF)

=> DATA FILE stores data and LOG FILE stores operations

NAME		TYPE	INITIAL SIZE	AUTOGROWTH	PATH
BATCH12	DATA	8	64	C:\	
BATCH12_LOG	LOG	8	64	C:\	

USER----SSMS-----SQL SERVER -----BATCH12(DB)

SQL :-

-----

=> STRUCTURED QUERY LANGUAGE

=> a language used to communicate with sql server.

=> user communicates with sql server by sending commands called queries.

=> a query is a command / instruction / question submitted to sql server to perform some operation over db.

=> SQL is originally introduced by IBM and initial name of this lang was SEQUEL and later it is renamed to SQL.

=> SQL is common to all RDBMS

sql server	oracle	mysql	postgresql
SQL	SQL	SQL	SQL

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

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

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

5-JUL-23

=> based on operations over db sql is divided into 5 sublanguages

DDL (DATA DEFINITION LANG)

DML (DATA MANIPULATION LANG)

DQL (DATA QUERY LANG)

TCL (TRANSACTION CONTROL LANG)

DCL (DATA CONTROL LANG)

SQL					
DDL	DML	DQL	TCL	DCL	
CREATE		INSERT	SELECT	COMMIT	GRANT
ALTER	UPDATE		ROLLBACK	REVOKE	

**DROP  
TRUNCATE**

**DELETE  
MERGE**

**SAVE TRANSACTION**

## **DATA & DATA DEFINITION :-**

-----

<b>EMPID</b>	<b>ENAME</b>	<b>SAL</b>	<b>DATA DEFINITION / METADATA</b>
<b>100</b>	<b>A</b>	<b>5000</b>	<b>DATA</b>

## **Datatypes in SQL SERVER :-**

-----

**=> a datatype specifies**

**1 type of the data allowed in column**

**2 amount of memory allocated for column**

## **character types :-**

-----

**ASCII**

**UNICODE**

**char**

**nchar**

**varchar**

**nvarchar**

**varchar(max)**

**nvarchar(max)**

## **char(size) :-**

-----

**=> allows character data upto 8000 chars**

**=> recommended for fixed length char columns**

**ex :-    NAME    CHAR(10)**

**SACHIN----**

**wasted**

**RAVI-----**

**wasted**

**NOTE :- in char datatype extra bytes are wasted , so char is not recommended for variable length columns and it is recommended for fixed length columns**

**STATE\_CODE    CHAR(2)**

**AP**

**TS**

**MH**

**COUNTRY\_CODE    CHAR(3)**

**IND**

## USA

### **VARCHAR(SIZE) :-**

-----

=> allows character data upto 8000 chars

=> recommended for variable length fields

ex :-    NAME    VARCHAR(10)

          SACHIN----  
              released

### **NOTE :-**

char/varchar allows ascii characters (256 chars) that includes a-z,A-Z,0-9 and special chars. so char/varchar allows alphanumeric data.

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

### **VARCHAR(MAX) :-**

-----

=> allows character data upto 2GB

ex :-    FEEDBACK    VARCHAR(MAX)

### **NCHAR/NVARCHAR/NVARCHAR(MAX) :-**

-----

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

### **Integer Types :-**

-----

=> allows numbers without decimal

<b>TINYINT</b>	<b>1 BYTE</b>	<b>0 TO 255</b>
<b>SMALLINT</b>	<b>2 BYTES</b>	<b>-32768 TO 32767</b>
<b>INT</b>	<b>4 BYTES</b>	<b>-2<sup>31</sup> TO 2<sup>31</sup>-1   (-2,147,483,647 to 2,147,483,646)</b>
<b>BIGINT</b>	<b>8 BYTES</b>	<b>-2<sup>63</sup> TO 2<sup>63</sup>-1   (-9,223,372,036,854,775,807 to 9,223,372,036,854,775,806)</b>

ex :-    AGE        TINYINT  
          EMPID     SMALLINT  
          ACCNO     BIGINT

### **NUMERIC(P) :-**

-----



=> allows numbers upto 38 digits

ex :- **EMPID NUMERIC(4)**

**10**

**100**

**1000**

**10000 => NOT ALLOWED**

**ACCNO NUMERIC(13)**

**AADHARNO NUMERIC(12)**

**CARD\_NO NUMERIC(16)**

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

=> allows numbers with decimal (float)

**p => precision => total no of digits allowed**

**s => scale => no of digits allowed after decimal**

ex :- **SAL NUMERIC(7,2)**

**5000**

**5000.55**

**50000.55**

**500000.55 => NOT ALLOWED**

**CURRENCY TYPES :-**  
-----

=> currency types are used for fields related to money

**SMALLMONEY 4 BYTES -214748.3648 to 214748.3647**

**MONEY 8 BYTES -922337203685477.5808  
to  
922337203685477.5807)**

**EX :- SALARY SMALLMONEY  
BALANCE MONEY**

**DATE & TIME :-**  
-----

**1 DATE => allows only date**

**2 TIME => allows only time**

**3 DATETIME => allows date & time**

=> default date format in sql server YYYY-MM-DD

=> default time format is HH:MI:SS

**EX :-**

**DOB      DATE**

**2003-04-20**

**LOGIN      TIME**

**9:30:00**

**TXN\_DT DATETIME**

**2023-07-05 10:00:00**

**06-jul-23**

### **CREATING TABLES IN DATABASE :-**

-----

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

### **Rules :-**

-----

**1 tabname should start with alphabet**

**2 tabname should not contain spaces & special chars but allows \_,#,\$**

**3 tabname can be upto 128 chars**

**4 table can have 1024 cols**

**5 no of rows unlimited**

<b>123cust</b>	<b>invalid</b>
<b>cust 123</b>	<b>invalid</b>
<b>cust*123</b>	<b>invalid</b>
<b>cust_123</b>	<b>valid</b>

### **Example :-**

**=> create table with following structure**

**EMP**  
**EMPID   ENAME   JOB   SAL   HIREDATE   DNAME**

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

)

=> above command created table structure (columns)

inserting data into table :-

-----

=> "insert" command is used to insert data into table.

=> we can insert

1 single row

2 multiple rows

inserting single row :-

-----

**INSERT INTO <tablename> VALUES(v1,v2,v3,-----)**

Ex :-

**INSERT INTO EMP VALUES(100,'SACHIN','CLERK',4000,'2023-07-06','HR')**

**INSERT INTO EMP VALUES(101,'ARVIND','MANAGER',8000,'2020-10-5','IT')**

inserting multiple rows :-

-----

**INSERT INTO EMP VALUES(102,'VIJAY','CLERK',6000,'2019-05-10','HR') ,  
(103,'RAVI','ANALYST',7000,'2018-02-15','SALES')**

inserting nulls :-

-----

=> a nulls means blank or empty

=> it is not equal to 0 or space

=> nulls can be inserted in two ways

method 1 :-

-----

**INSERT INTO EMP VALUES(104,'KUMAR',NULL,NULL,'2021-04-12','IT')**

method 2 :-

**INSERT INTO EMP(EMPID,ENAME,HIREDATE,DNAME)**

**VALUES(105,'SATISH','2022-09-10','SALES')**

remaining two fields job,sal filled with NULLs.

Operators in sql server :-

-----

**1 Arithmetic Operators => + - \* / %**

**2 Relational Operators => > >= < <= = <> or !=**

**3 Logical Operartors    =>    AND    OR    NOT**

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

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

### **Displaying Data :-**

-----

**=> "SELECT" command is used to display data from table.**

**=> we can display all rows and all columns**

**=> we can display specific rows and specific columns**

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

**SQL        =        ENGLISH**

**QUERIES =    SENTENCES**

**CLAUSES =    WORDS**

**\*    => 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 and hiredates ?**

**SELECT ENAME,HIREDATE FROM EMP**

### **WHERE clause :-**

-----

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

**SELECT columns  
FROM tablename  
WHERE condition**

**condition :-**  
-----

**COLNAME OP VALUE**

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

**=> if cond = true row is selected**

**=> if cond = false row is not selected**

**=> display employee details whose id = 103 ?**

**SELECT \* FROM EMP WHERE EMPID = 103**

**SELECT \* FROM EMP WHERE ENAME='KUMAR'**

**SELECT \* FROM EMP WHERE SAL>5000**

**SELECT \* FROM EMP WHERE HIREDATE > 2020 => ERROR**

**SELECT \* FROM EMP WHERE HIREDATE > '2020-12-31'**

**SELECT \* FROM EMP WHERE HIREDATE < '2020-01-01'**

**SELECT \* FROM EMP WHERE DNAME <> 'HR'**

**Compound condition :-**  
-----

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

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

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

**=> display employees whose id = 100,103,105 ?**

**SELECT \* FROM EMP WHERE EMPID=100 OR EMPID=103 OR EMPID=105**

**=> display employees working as CLERK,MANAGER ?**

**SELECT \* FROM EMP WHERE JOB='CLERK' OR JOB='MANAGER'**

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

**SELECT \* FROM EMP WHERE SAL>5000 AND SAL<10000**

**=> employees joined in 2020 ?**

**SELECT \*  
FROM EMP  
WHERE HIREDATE >= '2020-01-01' AND HIREDATE <= '2020-12-31'**

=> employees working as CLERK and earning more than 5000 and working for HR dept ?

```
SELECT *  
FROM EMP  
WHERE JOB='CLERK' AND SAL>5000 AND DNAME='HR'
```

**IN operator :-**  
-----

=> use IN operator for list comparison

=> use IN operator for "=" comparison with multiple values

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

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

=> employees working for HR,IT depts ?

```
SELECT * FROM EMP WHERE DNAME='HR' OR DNAME='IT'
```

```
SELECT * FROM EMP WHERE DNAME IN ('HR','IT')
```

=> employees not working as CLERK,MANAGER ?

```
SELECT * FROM EMP WHERE JOB NOT IN ('CLERK','MANAGER')
```

**BETWEEN operator :-**  
-----

=> use BETWEEN operator for range comparison

```
WHERE COLNAME BETWEEN V1 AND V2  
WHERE COLNAME NOT BETWEEN V1 AND V2
```

=> display 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'
```

=> employees working as CLERK,MANAGER and earning between 5000 and 10000 and joined in 2020 year and not working for HR,SALES dept ?

```
SELECT *  
FROM EMP  
WHERE JOB IN ('CLERK','MANAGER')  
AND  
SAL BETWEEN 5000 AND 10000
```

**AND  
HIREDATE BETWEEN '2020-01-01' AND '2020-12-31'  
AND  
DNAME NOT IN ('HR','SALES')**

**=> list of samsung,redmi,oneplus mobile phones price between 10000 and 20000 ?**

**PRODUCTS**

**prodid   pname   price   category   brand**

**SELECT \*  
FROM PRODUCTS  
WHERE CATEGORY='MOBILES'  
AND  
BRAND IN ('SAMSUNG','REDMI','ONEPLUS')  
AND  
PRICE BETWEEN 10000 AND 20000**

**=> list of male customers age between 20 and 30 and staying hyd,mum,blr ?**

**CUST**

**CUSTID   NAME   AGE   CITY   GENDER**

**SELECT \*  
FROM CUST  
WHERE GENDER='M'  
AND  
AGE BETWEEN 20 AND 30  
AND  
CITY IN ('HYD','MUM','BLR')**

**08-JUL-23**

**LIKE operator :-**

-----

**=> use LIKE operator for pattern comparision**

**ex :-   name starts with 'S'  
         emailid ends with '.in'**

**WHERE COLNAME LIKE 'PATTERN'  
WHERE COLNAME NOT LIKE 'PATTERN'**

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

**wildcard chars :-**

-----

**%       => 0 or many chars**

**\_       => exactly 1 char**

**=> employees name starts with 'S' ?**

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

**=> name ends with 'S' ?**

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

**=> where 'A' is the 4th char in their name ?**

**SELECT \* FROM EMP WHERE ENAME LIKE '\_\_\_A%'**

**=> 'A' is the 2nd char from last ?**

**SELECT \* FROM EMP WHERE ENAME LIKE '%A\_'**

**=> name contains 4 chars ?**

**SELECT \* FROM EMP WHERE ENAME LIKE '\_\_\_\_'**

**=> list of employees joined in oct month ?**

**YYYY-MM-DD**

**SELECT \* FROM EMP WHERE HIREDATE LIKE '\_\_\_\_10\_\_'**

**=> employees joined in 2020 year ?**

**SELECT \* FROM EMP WHERE HIREDATE LIKE '2020%'**

**=> display employees name starts with 'A','K','R' ?**

**SELECT \* FROM EMP WHERE ENAME LIKE 'A%'**  
**OR**  
**ENAME LIKE 'K%'**  
**OR**  
**ENAME LIKE 'R%'**

**SELECT \* FROM EMP WHERE ENAME LIKE '[AKR]%'**

**=> employees name starts between 'A' and 'P' ?**

**SELECT \* FROM EMP WHERE ENAME LIKE '[A-P]%'**

**IS operator :-**

-----

**=> use IS operator for NULL comparison**

**WHERE COLNAME IS NULL**  
**WHERE COLNAME IS NOT NULL**



**=>: employees not earning salary ?**

**SELECT \* FROM EMP WHERE SAL IS NULL**

**=> employees earning salary ?**

**SELECT \* FROM EMP WHERE SAL IS NOT NULL**

**summary :-**

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

**WHERE COLNAME BETWEEN V1 AND V2**

**WHERE COLNAME LIKE 'PATTERN'**

**WHERE COLNAME IS NULL**

**Question :-**

**1**

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

**A ERROR**

**B RETURNS CLERK & MANAGER**

**C RETURNS ONLY CLERK**

**D NONE**

**ANS :- C**

**2 SELECT \* FROM EMP WHERE JOB = 'CLERK' OR JOB LIKE 'MAN%'**

**ANS :- B**

**3 SELECT \* FROM EMP WHERE SAL BETWEEN 5000 AND 2000**

**A ERROR**

**B RETURNS ROWS**

**C RETURNS NO ROWS**

**D NONE**

**ANS :- C**

**WHERE SAL BETWEEN 2000 AND 5000 (SAL>=2000 AND SAL<=5000)**

**WHERE SAL BETWEEN 5000 AND 2000 (SAL>=5000 AND SAL<=2000)**

**ALIAS :-**

-----

**=> alias means another name or alternative name**

**=> used to change column heading**

**syn :- COLNAM / EXPR [AS] ALIAS**

**=> display ENAME ANNUAL SALARY ?**

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

**SELECT ENAME,SAL\*12 AS [ANNUAL SAL] 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**

<b>SACHIN</b>	<b>4000</b>	<b>800</b>	<b>1200</b>	<b>400</b>	<b>5600</b>
---------------	-------------	------------	-------------	------------	-------------

**ORDER BY clause :-**

-----

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

**SELECT columns  
FROM tablename  
[WHERE cond]  
ORDER BY colname ASC/DESC**

**=> default order is ASC**

**=> arrange employee list name wise asc order ?**

**SELECT \*  
FROM EMP  
ORDER BY ENAME ASC**

**=> arrange sal wise desc order ?**

**SELECT \*  
FROM EMP  
ORDER BY SAL DESC**

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

**SELECT ENAME,SAL,DNAME  
FROM EMP  
ORDER BY DNAME ASC,SAL DESC**

<b>1</b>	<b>A</b>	<b>3000</b>	<b>HR</b>
<b>2</b>	<b>B</b>	<b>5000</b>	<b>SALES</b>

<b>5</b>	<b>E</b>	<b>6000</b>	<b>HR</b>
<b>1</b>	<b>A</b>	<b>3000</b>	<b>HR</b>

3 C 4000 IT	=====>	6 F 5000 IT
4 D 2000 SALES		3 C 4000 IT
5 E 6000 HR		2 B 5000 SALES
6 F 5000 IT		4 D 2000 SALES

=> arrange list dept wise asc and with in dept hiredate wise asc ?

```
SELECT ENAME,SAL,HIREDATE,DNAME
FROM EMP
ORDER BY DNAME ASC ,HIREDATE ASC
```

scenario :-

-----

#### STUDENTS

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

=> arrange student list avg wise desc , m desc,p desc ?

```
SELECT * , (M+P+C)/3 AS AVG
FROM STUDENTS
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	50	70

=> display students list along with avg who got distinction ?

```
SELECT * , (M+P+C)/3 AS AVG
FROM STUDENTS
WHERE (M+P+C)/3 >= 70
ORDER BY (M+P+C)/3 DESC,M DESC,P DESC
```

DISTINCT clause :-

-----

=> eliminates duplicates from the select statement output.

```
SELECT DISTINCT colname
```

Ex :-

```
SELECT DISTINCT DNAME FROM EMP
```

```
HR
IT
SALES
```

**SELECT DISTINCT JOB FROM EMP**

**ANALYST  
CLERK  
MANAGER**

**TOP clause :-**  
-----

**=> used to find top n rows**

**syn :- SELECT TOP <n> COLNAMES / \***

**examples :-**

**=> display first 3 rows from emp table ?**

**SELECT TOP 3 \* FROM EMP**

**=> display top 3 highest paid employees ?**

**SELECT TOP 3 \*  
FROM EMP  
ORDER BY SAL DESC**

**=> display top 3 employees based on experience ?**

**SELECT TOP 3 \*  
FROM EMP  
ORDER BY HIREDATE ASC**

**=> display top 3 max salaries ?**

**SELECT TOP 3 SAL  
FROM EMP  
ORDER BY SAL DESC**

**summary :-**

<b>WHERE</b>	<b>=&gt; to select specific rows</b>
<b>ORDER BY</b>	<b>=&gt; to sort rows</b>
<b>DISTINCT</b>	<b>=&gt; to eliminate duplicates</b>
<b>TOP</b>	<b>=&gt; to select top n rows</b>

**DML commands :- (Data Manipulation Lang)**  
-----

**INSERT  
UPDATE  
DELETE  
MERGE**

**=> all DML commands acts on table data.**

11-jul-23

## **UPDATE :-**

-----

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

**syn :-**

```
UPDATE <TABNAME>  
SET COLNAME = VALUE , COLNAME = VALUE , -----  
[WHERE CONDITION]
```

**Ex :-**

- => update all employees comm with 500 ?**

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

**NOTE :-**

- => in SQL SERVER operations are auto committed (saved)**
- => to stop auto commit execute the following command**

```
SET IMPLICIT_TRANSACTIONS ON
```

- => after executing above command operations are not automatically committed**
- => to save the operation execute commit.**
- => to cancel the operation execute rollback.**

- => update employees comm with 800 whose job is salesman and joined in 1981 year ?**

```
UPDATE EMP  
SET COMM = 800  
WHERE JOB='SALESMAN'  
AND  
HIREDATE LIKE '1981%'
```

- => update sal with 1000 and comm with 800 whose empno = 7369 ?**

```
UPDATE EMP  
SET SAL = 1000 , COMM = 800  
WHERE EMPNO = 7369
```

- => increment salaries by 20% and comm by 10% those working as CLERK,MANAGER ?**

```
UPDATE EMP  
SET SAL = SAL + (SAL*0.2) , COMM = COMM + (COMM*0.1)  
WHERE JOB IN ('CLERK','MANAGER')
```

- => transfer employees from 10th dept to 30th dept ?**

**UPDATE EMP  
SET DEPTNO = 30  
WHERE DEPTNO = 10**

**scenario :-**

**PRODUCTS**  
**prodid pname price category brand**

**=> increase samsung,oneplus,realme mobile phones price by 10% ?**

**UPDATE PRODUCTS  
SET PRICE = PRICE + (PRICE\*0.1)  
WHERE BRAND IN ('SAMSUNG','ONEPLUS','REALME')  
AND  
CATEGORY='MOBILES'**

**DELETE command :-**  
-----

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

**syn :- DELETE FROM <TABNAME> [WHERE COND]**

**ex :-**

**=> delete all rows from emp table ?**

**DELETE FROM EMP**

**=> delete employees whose id = 7369 , 7566,7844 ?**

**DELETE FROM EMP WHERE EMPNO IN (7369,7566,7844)**

**DDL commands :- (Data Definition Lang)**  
-----

**CREATE  
ALTER  
DROP  
TRUNCATE**

**=> all DDL commands acts on table structure ( columns,datatype and size).**

**ALTER command :-**  
-----

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

**1 add columns  
2 drop columns**

**3 modify a column  
changing datatype  
changing size**

**Adding column :-**  
-----

**ex :- add column gender to emp table ?**

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

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

**UPDATE EMP SET GENDER='M' WHERE EMPNO = 7369**

**Dropping column :-**  
-----

**=> drop columns gender,comm from emp table ?**

**ALTER TABLE EMP  
DROP COLUMN GENDER,COMM**

**Modifying a column :-**  
-----

**=> modify the empno column datatype to int ?**

**ALTER TABLE EMP  
ALTER COLUMN EMPNO INT**

**=> increase size of ename to 20 ?**

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

**ALTER TABLE EMP  
ALTER COLUMN ENAME VARCHAR(5) => ERROR =>**

**some names contains more than 5 chars**

**12-JUL-23**

**DROP command :-**  
-----

**=> command used to drop table from db  
=> drops table structure along with data**

**syn :- DROP TABLE <tablename>**

**ex :- DROP TABLE STUDENTS**

## **TRUNCATE command :-**

- => deletes all data from table but keeps structure
- => will empty the table.
- => releases memory allocated for table,

**syn :- TRUNCATE TABLE <tablename>**

**Ex :- TRUNCATE TABLE EMP**

## **DROP VS DELETE VS TRUNCATE :-**

### **DROP**

**drops structure along with data**

### **DELETE/TRUNCATE**

**deletes only data but not structure**

## **DELETE VS TRUNCATE :-**

### **DELETE**

### **TRUNCATE**

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

## **SP\_RENAME :- ( SP -> stored procedure)**

- => used to change table name or column name

**SP\_RENAME ' OLD NAME ', ' NEW NAME '**

**ex :-**

- => rename table emp to employees ?

**SP\_RENAME 'EMP','EMPLOYEES'**



=> rename column comm to bonus ?

**SP\_RENAME 'EMPLOYEES.COMM','BONUS'**

## **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 NUMERIC**
- 4 CONVERSION**
- 5 SPECIAL**
- 6 ANALYTICAL**
- 7 AGGREGATE**

### **DATE functions :-**

-----

#### **1 GETDATE() :-**

-----

=> returns current date & time

```
SELECT GETDATE() => 2023-07-12 12:03:08.503
                   -----
                   DATE      TIME      MS
```

#### **2 DATEPART() :-**

-----

=> used to extract part of the date

**DATEPART(interval,date)**

**ex :-**

```
SELECT  DATEPART(YY,GETDATE()) => 2023
        MM                07
        DD                12
        DW                4 (wed)
        DY                193 (day of year)
        HH                hour part
        MI                minutes
        SS                seconds
        Q                 3
                        jan-mar 1
                        apr-jun 2
                        jul-sep 3
```

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

```
SELECT *
FROM EMP
WHERE DATEPART(YEAR,HIREDATE) IN (1980,1983,1985)
```

=> employees joined in leap year ?

```
SELECT *
FROM EMP
WHERE DATEPART(YEAR,HIREDATE)%4 = 0
```

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

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

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

```
SELECT *
FROM EMP
WHERE DATEPART(YEAR,HIREDATE) = 1981
AND
DATEPART(QUARTER,HIREDATE) = 2
```

**DATENAME() :-**

-----

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

	MM	DW
DATEPART	7	4
DATENAME	JULY	WEDNESDAY

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

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

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

```
SELECT ENAME + ' joined on ' + DATENAME(DW,HIREDATE)
FROM EMP
```

13-JUL-23

**DATEDIFF() :-**

-----

=> returns difference between two dates in given interval

**DATEDIFF(INTERVAL,START DATE,END DATE)**

**EX :-**

```
SELECT DATEDIFF(YY,'2022-07-13',GETDATE()) => 1
        MM                                     => 12
        DD                                     => 365
```

=> display ENAME EXPERIENCE in years ?

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

=> display ENAME EXPERIENCE ?  
M years N months

experience = 40 months = 3 years 4 months

years = months/12 = 40/12 = 3

months = months%12 = 40%12 = 4

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

**FORMAT() :-**

-----

=> function used to display dates in different formats

**FORMAT(DATE,'format')**

**ex :-**

```
SELECT FORMAT(GETDATE(),'MM/dd/yy')           => 07/13/23
SELECT FORMAT(GETDATE(),'dd.MM.yyyy')         => 13.07.2023
SELECT FORMAT(GETDATE(),'dd.MM.yyyy hh:mm')    => 13.07.2023 11:46
```

```
SELECT ENAME,FORMAT(HIREDATE,'MM/dd/yy') AS HIREDATE FROM EMP
```

**scenario :-**

-----

```
INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,HIREDATE)
VALUES(999,'ABC','CLERK',5000,GETDATE())
```

=> list of employees joined today ?

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

2023-07-13 = 2023-07-13 11:58:20.123

=> "=" comparison with getdate() always fails , to overcome this problem use format function

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

2023-07-13 = 2023-07-13

**DATEADD() :-**

-----

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

**DATEADD(INTERVAL,INT,DATE)**

```
SELECT DATEADD(DD,10,GETDATE()) => 2023-07-23  
SELECT DATEADD(MM,2,GETDATE()) => 2023-09-13  
SELECT DATEADD(MM,-2,GETDATE()) => 2023-05-13
```

**scenario :-**

-----

**GOLD\_RATES**

DATEID	RATE
--------	------

2020-01-01	?
------------	---

2020-01-02	?
------------	---

2023-07-13	?
------------	---

1 display today's gold rate ?

2 display yesterday's gold rate ?

3 SELECT \*

FROM GOLD\_RATES

WHERE DATEID = FORMAT(DATEADD(DD,-1,GETDATE()),'yyyy-MM-dd')

4 display last year same day gold rate ?

1

SELECT \*

FROM GOLD\_RATES

WHERE DATEID = FORMAT(GETDATE(),'yyyy-MM-dd')

2

SELECT \*

FROM GOLD\_RATES

**WHERE DATEID =   FORMAT(DATEADD(DD,-1,GETDATE()),'yyyy-MM-dd')**

**3**

**SELECT \*  
FROM GOLD\_RATES  
WHERE DATEID =   FORMAT(DATEADD(MM,-1,GETDATE()),'yyyy-MM-dd')**

**4**

**SELECT \*  
FROM GOLD\_RATES  
WHERE DATEID =   FORMAT(DATEADD(YY,-1,GETDATE()),'yyyy-MM-dd')**

**5 display last 1 month gold rates ?**

**2023-06-13   ?**

**2023-07-13   ?**

**SELECT \*  
FROM GOLD\_RATES  
WHERE DATEID BETWEEN  
                    FORMAT(DATEADD(MM,-1,GETDATE()),'yyyy-MM-dd')  
                    AND  
                    FORMAT(GETDATE(),'yyyy-MM-dd')**

**EOMONTH() :-**

-----

**=> returns last day of the month**

**EOMONTH(DATE,INT)**

**SELECT EOMONTH(GETDATE(),0) => 2023-07-31  
SELECT EOMONTH(GETDATE(),1) => 2023-08-31  
SELECT EOMONTH(GETDATE(),-1) => 2023-06-30**

**=> display next month 1st day ?**

**=> display current month 1st day ?**

**=> display next year 1st day ?**

**=> display current year 1st day ?**

**STRING fuctions :-**

-----

**UPPER() :-**

-----

**=> converts string to uppercase**

**UPPER(string)**

**ex :-**

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

**LOWER() :-**  
-----

**=> converts string to lowercase**

**LOWER(string)**

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

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

**SELECT EMPNO,LOWER(ENAME) AS ENAME,SAL FROM EMP**

**=> convert names to lowercase in table ?**

**update emp set ename = lower(ename)**

**14-jul-23**  
-----

**LEN() :-**  
-----

**=> returns string length i.e. no of characters**

**LEN(string)**

**ex :-**

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

**SELECT EMPNO,ENAME,LEN(ENAME) AS LEN FROM EMP**

**=> display employees name contains 5 chars ?**

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

**LEFT() :-**  
-----

**=> returns character starting from left**

**LEFT(string,len)**

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

**=> employees name starts with 's' ?**

**WHERE ENAME LIKE 's%'**

**SELECT \* FROM EMP WHERE LEFT(ENAME,1) = 's'**

**=> generate emailids for employees ?**

<b>empno</b>	<b>ename</b>	<b>emailid</b>
<b>7369</b>	<b>smith</b>	<b>smi736@tcs.com</b>
<b>7499</b>	<b>allen</b>	<b>all749@tcs.com</b>

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

**=> store emailids in db ?**

**step 1 :- add emailid column to emp table**

**ALTER TABLE EMP  
ADD EMAILID VARCHAR(30) ;**

**step 2 :- update the column with emailids**

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

**RIGHT() :-**  
-----

**=> returns character starting from right side**

**RIGHT(String,Len)**

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

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

**SELECT \*  
FROM EMP  
WHERE LEFT(ENAME,1) = RIGHT(ENAME,1)**

**SUBSTRING() :-**  
-----

**=> returns characters starting from specific position**

**SUBSTRING(string,start,len)**

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

**REPLICATE() :-**  
-----

**=> repeats character for given no of times**

**REPLICATE(char,len)**

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

**display   ENAME       SAL   ?**  
              \*\*\*\*

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

**SMITH   \*\*\*\*\***  
**ALLEN   \*\*\*\*\***

**=>**

**ACCOUNTS**  
**ACCNO               PHONE**  
**123456789573       9876543292**

**1   your a/c no XXXX9573 debited ----**

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

**2   display phone as 98XXXXXX892**

**LEFT(PHONE,2) + REPLICATE('X',5) + RIGHT(PHONE,3)**

**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('@@he@@ll@@o@@@','@','') => hello**

**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**



## NOTE :-

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

```
SELECT ENAME,  
       TRANSLATE(SAL,'0123456789.' , '$KT*b^%&@#!') as SAL  
FROM EMP
```

JONES 2975.00 T#&^!\$\$

15-jul-23

## CHARINDEX() :-

-----

=> returns position of a character in string.

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

ex :-

```
SELECT CHARINDEX('O','HELLO WELCOME')      => 5  
SELECT CHARINDEX('X','HELLO WELCOME')      => 0  
SELECT CHARINDEX('O','HELLO WELCOME',6)    => 11  
SELECT CHARINDEX('E','HELLO WELCOME',10)   => 13
```

## Assignment :-

```
CUST  
CID  CNAME  
10   SACHIN TENDULKAR  
11   VIRAT KOHLI
```

=> display            CID   FNAME   LNAME            ?  
                     10   SACHIN   TENDULKAR

using :- SUBSTRING , CHARINDEX

## STUFF() :-

-----

=> similar to replace used to replace a string based on start and length

**STUFF(string1,start,len,string2)**

```
SELECT STUFF('hello welcome',10,4,'abc')  => hello welabc  
SELECT STUFF('a,b,c,d,',8,1,"")           => a,b,c,d
```

## Numeric functions :-

-----

## rounding numbers :-

-----

**ROUND**  
**FLOOR**  
**CEILING**

**38.45678955 => 38**  
**38.45**  
**38.4567**

**ROUND() :-**  
-----

**=> rounds number to integer or to decimal places based on avg.**

**ROUND(number,decimal places)**

**ex :-**

**SELECT ROUND(38.4567,0)      => 38**

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

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

**SELECT ROUND(38.5567,0)      => 39**

**SELECT ROUND(38.4567,2)      => 38.46**

**SELECT ROUND(38.4537,2)      => 38.45**

**SELECT ROUND(386,-2)          => 400**

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

**SELECT ROUND(386,-1)          => 390**

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

**SELECT ROUND(386,-3)          => 0**

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

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

**O/P :- 4570    4600      5000**

**FLOOR() :-**

**=> always rounds number to lowest**

**FLOOR(number)**

**SELECT FLOOR(3.9)    => 3**

## **CEILING() :-**

-----

=> rounds number always to highest

**CEILING(number)**

**SELECT CEILING(3.1) => 4**

=> round employees salaries to hundreds ?

**UPDATE EMP SET SAL = ROUND(SAL,-2)**

## **conversion :-**

-----

=> used to convert one datatype to another datatype.

**1 CAST**

**2 CONVERT**

## **CAST :-**

-----

**CAST(source-value as target-type)**

**EX :-**

**SELECT CAST(10.5 AS INT) => 10**

**SELECT CAST(10 AS DECIMAL(5,3))=> 10.000**

=> display smith earns 800

allen earns 1600 ?

**SELECT ENAME + ' earns ' + CAST(SAL AS VARCHAR)  
FROM EMP**

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

**SELECT**

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

## **CONVERT() :-**

-----

**CONVERT(TARGET-TYPE,SOURCE-VALUE)**

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

## **special functions :-**

-----

## **ISNULL() :-**

-----

=> used to convert null values

**ISNULL(arg1,arg2)**

if arg1 = null returns arg2

if arg1 <> null returns arg1 only

**SELECT ISNULL(100,200) => 100**

**SELECT ISNULL(NULL,200) => 200**

**display ENAME SAL COMM TOTSAL?**

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

<b>SMITH</b>	<b>800</b>	<b>NULL</b>	<b>800</b>
<b>ALLEN</b>	<b>1600</b>	<b>300</b>	<b>1900</b>

=> display ENAME SAL COMM ?

if comm = NULL display NO COMM

**SELECT ENAME,SAL,  
ISNULL(CAST(COMM AS VARCHAR),'NO COMM') AS COMM  
FROM EMP**

**17-JUL-23**

## **Analytical Functions / Window Functions :-**

-----

### **RANK() & DENSE\_RANK() :-**

-----

=> both functions are used to find ranks

=> ranks are based on some column

=> for rank functions data must be sorted

**RANK() OVER (ORDER BY COLNAME ASC/DESC , -----)  
DENSE\_RANK() OVER (ORDER BY COLNAME ASC/DESC,---)**

**Examples :-**

=> find the ranks of the employees based on sal and highest paid 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**

**difference between rank & dense\_rank ?**

**1 rank function generates gaps but dense\_rank will not generate gaps**

**2 in rank function ranks may not be in sequence but in dense\_rank ranks are always in sequence**

<b>SAL</b>	<b>RNK</b>	<b>DRNK</b>
<b>5000</b>	<b>1</b>	<b>1</b>
<b>4000</b>	<b>2</b>	<b>2</b>
<b>3000</b>	<b>3</b>	<b>3</b>
<b>3000</b>	<b>3</b>	<b>3</b>
<b>3000</b>	<b>3</b>	<b>3</b>
<b>2000</b>	<b>6</b>	<b>4</b>
<b>2000</b>	<b>6</b>	<b>4</b>
<b>1000</b>	<b>8</b>	<b>5</b>

**=> find ranks of the employees based on sal , if salaries are same then ranking should be based on hiredate ?**

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

<b>king</b>	<b>1981-11-17</b>	<b>5000.00</b>	<b>1</b>
<b>abc</b>	<b>2023-07-13</b>	<b>5000.00</b>	<b>2</b>
<b>jones</b>	<b>1981-04-02</b>	<b>3000.00</b>	<b>3</b>
<b>ford</b>	<b>1981-12-03</b>	<b>3000.00</b>	<b>4</b>
<b>scott</b>	<b>1982-12-09</b>	<b>3000.00</b>	<b>5</b>
<b>blake</b>	<b>1981-05-01</b>	<b>2900.00</b>	<b>6</b>

**=>**

**STUDENT**

<b>SNO</b>	<b>SNAME</b>	<b>M</b>	<b>P</b>	<b>C</b>
<b>1</b>	<b>A</b>	<b>80</b>	<b>90</b>	<b>70</b>
<b>2</b>	<b>B</b>	<b>70</b>	<b>60</b>	<b>50</b>
<b>3</b>	<b>C</b>	<b>90</b>	<b>70</b>	<b>80</b>
<b>4</b>	<b>D</b>	<b>90</b>	<b>80</b>	<b>70</b>

**=> find ranks of the students based on total desc, m desc,p desc ?**

**PARTITION BY clause :-**

-----

**=> used to find ranks with in group , for ex to find ranks with in dept first divide the table dept wise and apply rank functions on each dept instead of applying it on whole table**

```
SELECT empno,ename,sal,deptno,  
       dense_rank() over (partition by deptno
```

**order by sal desc) as rnk**

**FROM emp**

**10**

<b>5000</b>	<b>1</b>
<b>2450</b>	<b>2</b>
<b>1300</b>	<b>3</b>

**20**

<b>3000</b>	<b>1</b>
<b>3000</b>	<b>1</b>
<b>2975</b>	<b>2</b>
<b>1100</b>	<b>3</b>
<b>800</b>	<b>4</b>

**ROW\_NUMBER() :-**

-----

**=> returns record numbers based on some column**

**=> data must be sorted**

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

<b>SAL</b>	<b>RNK</b>	<b>DRNK</b>	<b>RNO</b>
<b>5000</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>4000</b>	<b>2</b>	<b>2</b>	<b>2</b>
<b>3000</b>	<b>3</b>	<b>3</b>	<b>3</b>
<b>3000</b>	<b>3</b>	<b>3</b>	<b>4</b>
<b>3000</b>	<b>3</b>	<b>3</b>	<b>5</b>
<b>2000</b>	<b>6</b>	<b>4</b>	<b>6</b>
<b>2000</b>	<b>6</b>	<b>4</b>	<b>7</b>
<b>1000</b>	<b>8</b>	<b>5</b>	<b>8</b>

**Aggregate Functions / Multi-row functions :-**

-----

**=> these functions process multiple rows and returns one value**

**MAX() :-**

-----

**=> returns maximum value**

**MAX(arg)**

<b>SELECT MAX(SAL) FROM EMP</b>	<b>=&gt; 5000.00</b>
<b>SELECT MAX(HIREDATE) FROM EMP</b>	<b>=&gt; 1983-01-12</b>
<b>SELECT MAX(ENAME) FROM EMP</b>	<b>=&gt; ward</b>

**MIN() :-**

-----

=> returns minimum value

**MIN(arg)**

**SELECT MIN(SAL) FROM EMP                    => 800**

**SUM() :-**

-----

=> returns total

**SUM(arg)**

**SELECT SUM(SAL) FROM EMP                    => 29300.00**

=> round total sal to thousands ?

**SELECT ROUND(SUM(SAL) , -3) FROM EMP => 29000**

**29000-----29500-----30000**

=> after rounding display total sal with thousand seperator ?

**SELECT     CONVERT(VARCHAR , ROUND(SUM(SAL) , -3) , 1 ) AS TOTSAL  
FROM EMP**

**O/P :- 29,000.00**

=> calculate total sal including comm ?

**SELECT SUM(SAL+COMM) AS TOTSAL FROM EMP => 7900**

<b>SAL</b>	<b>COMM</b>	<b>SAL+COMM</b>
<b>5000</b>	<b>NULL</b>	<b>NULL</b>
<b>4000</b>	<b>500</b>	<b>4500</b>
<b>3000</b>	<b>NULL</b>	<b>NULL</b>

<b>SUM(SAL)</b>	<b>= 12000</b>
<b>SUM(SAL+COMM)</b>	<b>= 4500</b>

**SELECT SUM(SAL+ISNULL(COMM,0)) AS TOTSAL FROM EMP => 31500**

<b>SAL</b>	<b>COMM</b>	<b>SAL+ISNULL(COMM,0)</b>
<b>5000</b>	<b>NULL</b>	<b>5000</b>
<b>4000</b>	<b>500</b>	<b>4500</b>
<b>3000</b>	<b>NULL</b>	<b>3000</b>

**SUM(SAL) = 12000**

**SUM(SAL+ISNULL(COMM,0)) = 12500**

**AVG() :-**  
-----

**=> returns average value**

**AVG(arg)**

**SELECT AVG(SAL) FROM EMP => 2092.8571**

**=> round avg sal to highest integer**

**SELECT CEILING(AVG(SAL)) FROM EMP => 2093.00**

**18-JUL-23**

**COUNT(\*) :-**  
-----

**=> returns no of rows in a table.**

**SELECT COUNT(\*) FROM EMP**

**=> no of employees joined in 1981 year ?**

**SELECT COUNT(\*)  
FROM EMP  
WHERE DATEPART(YEAR,HIREDATE) = 1981**

**=> no of employees joined on sunday ?**

**SELECT COUNT(\*)  
FROM EMP  
WHERE DATENAME(dw,HIREDATE) = 'SUNDAY'**

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

**SELECT COUNT(\*)  
FROM EMP  
WHERE DATEPART(YEAR,HIREDATE)=1981  
AND  
DATEPART(QUARTER,HIREDATE) = 2**

**NOTE :-**

**=> aggregate functions are not allowed in where clause and they are allowed only in  
SELECT,HAVING clauses.**

**SELECT ENAME  
FROM EMP  
WHERE SAL = MAX(SAL) => ERROR**



**summary :-**

**DATE :-** datepart,datetime,datediff,dateadd,format,eomonth

**STRING :-** upper,lower,len,left,right,substring,replicate,replace,translate,stuff,charindex

**NUMERIC :-** round,floor,ceiling

**CONVERSION :-** cast,convert

**SPECIAL :-** isnull

**ANALYTICAL :-** rank,dense\_rank,row\_number

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

=====

**CASE statement :-**

-----

- => case statement is similar to switch case.
- => used to implement if-else in sql.
- => using case statement we can return values based on condition.
- => case statements are 2 types

- 1 simple case
- 2 searched case

**simple case :-**

-----

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

=> **DISPLAY ENAME JOB ?**

<b>IF JOB=CLERK</b>	<b>DISPLAY WORKER</b>
<b>MANAGER</b>	<b>BOSS</b>
<b>PRESIDENT</b>	<b>BIG BOSS</b>
<b>OTHERS</b>	<b>EXECUTIVE</b>

```
SELECT ENAME,
CASE JOB
WHEN 'CLERK' THEN 'WORKER'
WHEN 'MANAGER' THEN 'BOSS'
WHEN 'PRESIDENT' THEN 'BIG BOSS'
ELSE 'EXECUTIVE'
END AS JOB
FROM EMP
```

=> increment employee salaries as follows ?

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

**UPDATE EMP**

```
SET SAL = CASE DEPTNO
  WHEN 10 THEN SAL + (SAL*0.1)
  WHEN 20 THEN SAL + (SAL*0.15)
  WHEN 30 THEN SAL + (SAL*0.2)
  ELSE SAL + (SAL*0.05)
END
```

searched case :-

=> use searched case when conditions not based on "=" i.e. based on > < between operators

**CASE**

```
WHEN COND1 THEN RETURN EXPR1
WHEN COND2 THEN RETURN EXPR2
```

```
ELSE RETURN EXPR
END
```

=> display ENAME SAL SALRANGE ?

```
IF SAL > 3000 DISPLAY HISAL
SAL < 3000 DISPLAY LOSAL
SAL=3000          AVGSAL
```

```
SELECT ENAME,SAL,
CASE
  WHEN SAL>3000 THEN 'HISAL'
  WHEN SAL<3000 THEN 'LOSAL'
  ELSE 'AVGSAL'
END AS SALRANGE
FROM EMP
```

=> display SNO TOTAL AVG RESULT ?

**STUDENT**

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

```
SELECT SNO,
  S1+S2+S3 AS TOTAL,
  (S1+S2+S3)/3 AS AVG,
```

```

CASE
WHEN S1>=35 AND S2>=35 AND S3>=35 THEN 'PASS'
ELSE 'FAIL'
END AS RESULT
FROM STUDENT

```

19-JUL-23

**GROUP BY clause :-**

=> **GROUP BY** clause groups rows based on one or more columns to calculate min,max,sum,avg,count for each group. For ex to calculate total sal paid to each dept first we need to group rows based on dept and apply sum(sal) function on each dept instead of applying on whole table.

EMP						
EMPNO	ENAME	SAL	DEPTNO			
1	A	5000	10			
2	B	4000	20	<b>GROUP BY</b>	10	7000
3	C	3000	30	=====>	20	8000
4	D	2000	10		30	3000
5	E	4000	20			

**detailed data**

**summarized data**

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

**syn :-**

```

SELECT columns
FROM tablename
[WHERE cond]
GROUP BY col1,col2,---
[HAVING cond]
[ORDER BY colname 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

```

10	8800.00
----	---------

20	10900.00
30	9600.00

**FROM EMP :-**

-----

**EMP**

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	10
2	B	4000	20
3	C	3000	30
4	D	2000	10
5	E	4000	20

**GROUP BY DEPTNO :-**

-----

10	1	A	5000
	4	D	2000
20	2	B	4000
	5	E	4000
30	3	C	3000

**SELECT DEPTNO,SUM(SAL) AS TOTSAL :-**

-----

10	7000
20	8000
30	3000

**=> 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(YEAR,HIREDATE) AS YEAR,COUNT(*) AS CNT
FROM EMP
GROUP BY DATEPART(YEAR,HIREDATE)
```

**=> display day wise no of employees joined ?**

```
SELECT DATENAME(DW,HIREDATE) AS DAY,COUNT(*) AS CNT
FROM EMP
GROUP BY DATENAME(DW,HIREDATE)
```

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

```
SELECT DATENAME(MONTH,HIREDATE) AS MONTH,COUNT(*) AS CNT
FROM EMP
```

**WHERE DATEPART(YEAR,HIREDATE)=1981  
GROUP BY DATENAME(MONTH,HIREDATE)**

**=> find the departments having more than 3 employees ?**

**SELECT DEPTNO,COUNT(\*) AS CNT  
FROM EMP  
WHERE COUNT(\*) > 3  
GROUP BY DEPTNO           => ERROR**

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

**SELECT DEPTNO,COUNT(\*) AS CNT  
FROM EMP  
GROUP BY DEPTNO  
HAVING COUNT(\*) > 3**

**WHERE VS HAVING :-**

<b>WHERE</b>	<b>HAVING</b>
<b>1   selects specific rows</b>	<b>selects specific groups</b>
<b>2   conditions executed before group by</b>	<b>conditions executed after group by</b>
<b>3   use where clause if cond doesn't contain aggregate function</b>	<b>use having clause if cond contains aggregate function</b>

**=> find southern states having more than 5CR population ?**

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

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

**20-jul-23**

**=> display dept wise total salaries where deptno = 10,20 and sum(sal) > 10000 ?**

**select deptno,sum(sal)  
from emp  
where deptno in (10,20)  
group by deptno  
having sum(sal) > 10000**

**Grouping based on multiple columns :-**

=> display dept wise and with in dept job wise no of employees ?

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

10	CLERK	1	
	MANAGER	1	
	PRESIDENT	1	
20	ANALYST		2
	CLERK	2	
	MANAGER	1	
30	CLERK	1	
	MANAGER	2	
	SALESMAN	4	

=>

#### PERSONS

AADHARNO	NAME	GENDER	AGE	ADDR	CITY	STATE
----------	------	--------	-----	------	------	-------

display state wise and with in state gender wise population ?

```
SELECT STATE,GENDER,COUNT(*) AS CNT
FROM EMP
GROUP BY STATE,GENDER
ORDER BY STATE ASC
```

AP	MALE	?
	FEMALE	?

AR	MALE	?
	FEMALE	?

=> display duplicate records ?

#### EMP11

ENO	ENAME	SAL
-----	-------	-----

1	A	5000
2	B	6000
1	A	5000
2	B	6000
3	C	4000

```
SELECT ENO,ENAME,SAL
FROM EMP11
GROUP BY ENO,ENAME,SAL
HAVING COUNT(*) > 1
```

1	A	5000
2	B	6000

=====

## **INTEGRITY CONSTRAINTS**

-----

- => Integrity Constraints are rules to maintain Data Quality.**
- => used to prevent users from entering invalid data.**
- => used to enforce rules like min bal must be 1000.**
- => different integrity constraints in sql server**

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

**=> above constraints can be declared in two ways.**

- 1 COLUMN LEVEL**
- 2 TABLE LEVEL**

### **COLUMN LEVEL :-**

-----

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

#### **NOT NULL :-**

-----

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

**ex :-**

```
CREATE TABLE EMP15  
(  
    ENO INT,  
    ENAME VARCHAR(10) NOT NULL  
)
```

**INSERT INTO EMP15 VALUES(1,NULL) => ERROR**  
**INSERT INTO EMP15 VALUES(2,'B')**

#### **UNIQUE :-**

-----

**=> unique constraint doesn't accept duplicates**

**ex :-**

```
CREATE TABLE CUST
```

```
(
  CID INT ,
  CNAME VARCHAR(10),
  EMAILID VARCHAR(20) UNIQUE
)
```

```
INSERT INTO CUST VALUES(10,'A','abc@gmail.com')
INSERT INTO CUST VALUES(11,'B','abc@gmail.com') => ERROR
INSERT INTO CUST VALUES(12,'C',NULL)
INSERT INTO CUST VALUES(13,'D',NULL) => ERROR
```

**PRIMARY KEY :-**

-----

=> primary key doesn't accept duplicates and nulls.  
=> it is combination of unique & not null.  
=> in tables one column must be there to uniquely identify the records and that column must be declared with primary key.

**ex :-**

```
CREATE TABLE EMP16
(
  EMPID INT PRIMARY KEY,
  ENAME VARCHAR(10) NOT NULL
)
```

```
INSERT INTO EMP16 VALUES(100,'A')
INSERT INTO EMP16 VALUES(100,'B') => ERROR
INSERT INTO EMP16 VALUES(NULL,'A') => ERROR
```

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

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

**difference between UNIQUE & PRIMARY KEY ?**

	UNIQUE	PRIMARY KEY
1	allows one null	doesn't allow null
2	multiple columns can be declared with unique	only one column can be declared with primary key

**candidate key :-**

-----



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

ex :-

**VEHICLE**  
**VEHNO    VNAME    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**.

28-jul-23

## **JOINS**

-----

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

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

Example :-

**orders**

**customer**

ordid	orddt	deldt	cid		cid	cname	caddr
1000		10		10	A	HYD	
1001			11	11	B	HYD	
1002			12	12	C	HYD	

output :-

ordid	orddt	deldt	cname	caddr
1000		A	HYD	
1001		B	HYD	

**Types of Joins :-**

-----

1 inner join

2 outer join

    left join

    right join

    full join

3 non equi join

4 self join

## 5 cross / cartesian join

## inner join / equi join :-

=> inner join is performed between the tables sharing common field and name of the common field need not to be same and pk-fk relationship is not compulsory.

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

## join condition :-

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

ex :-

EMP				DEPT			
EMPNO	ENAME	SAL	DEPTNO	DEPTNO	DNAME	LOC	
1	A	5000	10	10	ACCTS	NEW YORK	
2	B	4000	30	20	RESEARCH		
3	C	2000	20	30	SALES		
4	D	3000	10	40	OPERATIONS		
5	E	2000	NULL				

=> display

ENAME	SAL	DNAME	LOC ?
EMP		DEPT	

```
SELECT ENAME,SAL,DNAME,LOC
FROM EMP INNER JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO
```

A	6000	ACCTS	NEW YORK
B	4000	SALES	???
C	2000	RESEARCH	???
D	3000	ACCTS	NEW YORK

=> in join queries declare table alias and prefix column names with table alias for two reasons

- 1 to avoid ambiguity
- 2 for faster execution

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

=> display employee names working at NEW YORK loc ?

```

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

```

joining more than two tables :-

-----

=> no of join conditions is based on no of tables to be joined.

=> to join N tables N-1 join conditions required.

syntax :-

```

SELECT COLUMNS
FROM TAB1 INNER JOIN T2
ON JOIN COND
INNER JOIN T3
ON JOIN COND
INNER JOIN T4
ON JOIN COND

```

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

```

SELECT E.ENAME,
       D.DNAME,
       L.CITY,L.STATE,
       C.COUNTRY_NAME
FROM EMP AS E INNER JOIN DEPT AS D
ON E.DEPTNO = D.DEPTNO
INNER JOIN LOCATIONS AS L
ON D.LOCID = L.LOCID
INNER JOIN COUNTRIES AS C
ON L.COUNTRY_ID = C.COUNTRY_ID

```

outer join :-

-----

=> inner join returns only matching records but cannot return unmatched records but to display unmatched records perform outer join.

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

=> outer join is 3 types

- 1 LEFT JOIN**
- 2 RIGHT JOIN**
- 3 FULL JOIN**

#### **LEFT JOIN :-**

-----

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

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

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

<b>A</b>	<b>ACCTS</b>
<b>B</b>	<b>SALES</b>
<b>C</b>	<b>RESEARCH</b>
<b>D</b>	<b>ACCTS</b>
<b>E</b>	<b>NULL =&gt; unmatched from emp</b>

#### **RIGHT JOIN :-**

-----

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

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

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

<b>A</b>	<b>ACCOUNTS</b>
<b>B</b>	<b>SALES</b>
<b>C</b>	<b>RESEARCH</b>
<b>D</b>	<b>ACCOUNTS</b>
<b>NULL</b>	<b>OPERATIONS =&gt; unmatched from dept</b>

#### **FULL JOIN :-**

-----

=> returns all rows from both tables

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

<b>A</b>	<b>ACCOUNTS</b>
<b>B</b>	<b>SALES</b>
<b>C</b>	<b>RESEARCH</b>
<b>D</b>	<b>ACCOUNTS</b>
<b>E</b>	<b>NULL =&gt; unmatched from emp</b>

**NULL OPERATIONS => unmatched from dept**

**Displaying only unmatched rows :-**

**left side table :-**

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

**E NULL**

**right side table :-**

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

**NULL OPERATIONS**

**both tables :-**

```
SELECT E.ENAME,D.DNAME
FROM EMP AS E FULL JOIN DEPT AS D
ON E.DEPTNO = D.DEPTNO
WHERE E.ENAME IS NULL
OR
D.DNAME IS NULL
```

**scenario :-**

<b>emp</b>			<b>projects</b>		
<b>empid</b>	<b>ename</b>	<b>projid</b>	<b>projid</b>	<b>pname</b>	<b>duration</b>
1	a	100		100	A
2	b	101		101	B
3	c	null		102	C

**=> display employee details with project details ?**

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

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

```
SELECT e.*,p.*
```

```
FROM emp as e LEFT JOIN projects as p
ON e.projid = p.projid
```

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

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

**NON EQUI JOIN :-**

-----

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

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

=> display ENAME                  GRADE ?

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

=> display employee names whose grade = 3 ?

```
SELECT E.ENAME,S.GRADE
FROM EMP AS E JOIN SALGRADE AS 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 AS E INNER JOIN DEPT AS D
ON E.DEPTNO = D.DEPTNO
JOIN SALGRADE S
ON E.SAL BETWEEN S.LOSAL AND S.HISAL
```

31-jul-23

**self join :-**

-----

=> joining a table to itself is called self join

=> in self join a record in one table joined with another record of same table.

ex :-

EMPNO	ENAME	MGR
7499	ALLEN	7698
7521	WARD	7698
7566	JONES	7839
7698	BLAKE	7839
7839	KING	NULL

=> above table contains manager number but to display manager name self join is required.

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

FROM EMP AS X JOIN EMP AS Y

EMP X			EMP Y		
EMPNO	ENAME	MGR	EMPNO	ENAME	MGR
7499	ALLEN	7698	7499	ALLEN	7698
7521	WARD	7698	7521	WARD	7698
7566	JONES	7839	7566	JONES	7839
7698	BLAKE	7839	7698	BLAKE	7839
7839	KING	NULL	7839	KING	NULL

=> display ENAME MGRNAME ?

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

ALLEN	BLAKE
WARD	BLAKE
JONES	KING
BLAKE	KING

=> display employees who are reporting to blake ?

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

=> display blake's manager name ?

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

=> employees earning more than or equal to their managers ?

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

=> employees joined before their managers ?

```
SELECT X.ENAME ,X.HIREDATE,
       Y.ENAME AS MGRNAME,Y.HIREDATE AS MGRHIRE
FROM EMP AS X JOIN EMP Y
      ON X.MGR = Y.EMPNO
WHERE X.HIREDATE < Y.HIREDATE
```

Question 1 :-

organizations

orgid	orgname	parent_org_id
100	TATA MOTORS HQ	NULL
101	TATA MOTORS USA	100
102	TATA MOTORS IND	100
103	TATA MOTORS NY	101
104	TATA MOTORS HYD	102

=> display orgname & parent org name ?

=> display orgnames reporting to tata motors ind ?

Question 2 :-

TEAMS

ID	COUNTRY
1	IND
2	AUS
3	ENG

=> write a query to display following output ?

IND VS AUS  
IND VS ENG  
AUS VS ENG

TEAMS A

ID	COUNTRY
1	IND
2	AUS
3	ENG

TEAMS B

ID	COUNTRY
1	IND
2	AUS
3	ENG

A.ID = B.ID      A.ID <> B.ID      A.ID < B.ID

IND IND	IND AUS	IND AUS
AUS AUS	IND ENG	IND ENG
ENG ENG	AUS IND	AUS ENG



**AUS ENG  
ENG IND  
ENG AUS**

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

**=> display ENAME DNAME GRADE MNAME ?**

**SELECT E.ENAME,D.DNAME,S.GRADE,M.ENAME  
FROM EMP AS E INNER JOIN DEPT AS D  
ON E.DEPTNO = D.DEPTNO  
JOIN SALGRADE AS S  
ON E.SAL BETWEEN S.LOSAL AND S.HISAL  
JOIN EMP AS M  
ON E.MGR = M.EMPNO**

**01-aug-23**

**cross join / cartesian join :-**  
-----

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

**A = 1,2  
B = 3,4**

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

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

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

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

**GROUP BY & JOIN :-**  
-----

**=> display dept wise total salary ? display dept names ?**

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

**ON e.deptno = d.deptno :-**  
-----

<b>EMP</b>				<b>DEPT</b>			
<b>EMPNO</b>	<b>ENAME</b>	<b>SAL</b>	<b>DEPTNO</b>		<b>DEPTNO</b>	<b>DNAME</b>	<b>LOC</b>
1	A	5000	10	10	ACCTS	NEW YORK	

2	B	4000	30	20	RESEARCH
3	C	2000	20	30	SALES
4	D	3000	10	40	OPERATIONS
5	E	2000	20		

A	5000	ACCTS
B	4000	SALES
C	2000	RESEARCH
D	3000	ACCTS
E	2000	RESEARCH

**GROUP BY d.dname :-**

**ACCTS**

A	5000
D	3000

**RESEARCH**

C	2000
E	2000

**SALES**

B	4000
---	------

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

ACCTS	8000
RESEARCH	4000
SALES	4000

**=> display no of employees working under each manager ?**

blake	?
king	?

**select y.ename as manager ,COUNT(x.ename) as cnt  
from emp as x join emp as y  
on x.mgr = y.empno  
group by y.ename**

**on x.mgr = y.empno**

EMP X		EMP Y			
EMPNO	ENAME MGR	EMPNO	ENAME	MGR	
7499	ALLEN	7698	7499	ALLEN	7698
7521	WARD	7698	7521	WARD	7698
7566	JONES	7839	7566	JONES	7839
7698	BLAKE	7839	7698	BLAKE	7839

7839 KING NULL 7839 KING NULL

X.ENAME Y.ENAME  
ALLEN BLAKE  
WARD BLAKE  
JONES KING  
BLAKE KING

GROUP BY Y.ENAME :-

-----  
  
BLAKE  
    ALLEN  
    WARD  
KING  
    JONES  
    BLAKE

select y.ename as manager ,COUNT(x.ename) as cnt :-

-----  
  
BLAKE 2  
KING 2

Assignment :-

-----  
  
SALES  
DATEID PRODID CUSTID QTY AMT  
2023-08-01 100 10 1 1000

PRODUCTS  
PRODID PNAME PRICE CATEGORY

CUST  
CUSTID CNAME ADDR COUNTRY

- 1 display category wise total amount ?
- 2 display country wise total amount ?
- 3 display year wise total amount ?
- 4 display year wise , country wise,category wise total amount ?

=====

SET OPERATORS :-

-----  
  
UNION  
UNION ALL  
INTERSECT  
EXCEPT

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

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

**=> in SQL , set operations performed between records return by two queries**

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

**Rules :-**

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

**UNION :-**  
 -----

- => combines rows return by two queries**
- => duplicates are eliminated**
- => result is sorted**

**SELECT job FROM emp WHERE deptno = 20**

**CLERK**  
**MANAGER**  
**ANALYST**  
**CLERK**  
**ANALYST**

**SELECT job FROM emp WHERE deptno = 30**

**SALESMAN**  
**SALESMAN**  
**SALESMAN**  
**MANAGER**  
**SALESMAN**  
**CLERK**

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

**ANALYST**  
**CLERK**  
**MANAGER**  
**SALESMAN**

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

ANALYST	3000.00
CLERK	800.00
CLERK	1000.00
CLERK	1100.00
MANAGER	2900.00
MANAGER	3000.00
SALESMAN	1300.00
SALESMAN	1500.00
SALESMAN	1600.00

## UNION VS JOIN :-

	UNION	JOIN
1	combines rows	combines columns
2	horizontal merge	vertical merge
3	performed between two queries	performed between two tables

T1	T2
F1	C1
1	10
2	20
3	30

T1 U T2	T1 JOIN T2
1	1      10
2	2      20
3	3      30
10	
20	
30	

## scenario :-

EMP_US	DEPT
ENO    ENAME   DNO	
	DNO    DNAME   LOC

=> total employees list ?

```
SELECT * FROM EMP_US
UNION
```

**SELECT \* FROM EMP\_IND**

**=> employees working in US with dept details ?**

**SELECT E.\*,D.\*  
FROM EMP\_US AS E INNER JOIN DEPT AS D  
ON E.DNO = D.DNO**

**=> total employees with dept details ?**

**SELECT E.\*,D.\*  
FROM EMP\_US AS E INNER JOIN DEPT AS D  
ON E.DNO = D.DNO  
UNION  
SELECT E.\*,D.\*  
FROM EMP\_IND AS E INNER JOIN DEPT AS D  
ON E.DNO = D.DNO**

**UNION ALL :-**

-----

**=> combines rows**

**=> duplicates are not eliminated**

**=> result is not sorted**

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

**CLERK  
MANAGER  
ANALYST  
CLERK  
ANALYST  
SALESMAN  
SALESMAN  
SALESMAN  
MANAGER  
SALESMAN  
CLERK**

**=> difference between UNION & UNION ALL ?**

	<b>UNION</b>	<b>UNION ALL</b>
<b>1</b>	<b>eliminates duplicates</b>	<b>duplicates are not eliminated</b>
<b>2</b>	<b>result is sorted</b>	<b>result is not sorted</b>
<b>3</b>	<b>slower</b>	<b>faster</b>

**INTERSECT :-**

-----

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

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

CLERK  
MANAGER

**EXCEPT :-**  
-----

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

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

ANALYST

**Question :-**  
-----

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

=> write outputs for the following operations ?

- 1 inner join
- 2 left join
- 3 right join
- 4 full join
- 5 union
- 6 union all
- 7 intersect
- 8 except

02-aug-23

**SUB-QUERIES / NESTED QUERIES :-**  
-----

- => a query in another query is called subquery or nested query.
- => one query is called inner / sub / nested query.
- => other query is called outer / main query.
- => first sql server executes inner query then it executes outer query.
- => result of inner query is input to outer query.
- => use subqueries when where cond based on unknown value.

## **Types of subqueries :-**

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

## **single row subqueries :-**

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

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

**=> op must be any relational operator like = >= <= <>**

**examples :-**

**=> employees earning more than blake ?**

```
SELECT *  
FROM EMP  
WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME='BLAKE')
```

**=> employees who are senior to king ?**

```
SELECT *  
FROM EMP  
WHERE HIREDATE < (SELECT HIREDATE FROM EMP WHERE ENAME='KING')
```

**=> name of the employee earning max salary ?**

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

**aggregate functions are not allowed in where clause and they are allowed in select, having clauses .**

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

-----  
**5000**

**=> name of the employee having max experience ?**

```
SELECT ename  
FROM emp
```



**WHERE hiredate = (SELECT MIN(hiredate) FROM emp)**

**=> 2nd max salary ?**

```
SELECT MAX(SAL)
FROM EMP
WHERE SAL <> (SELECT MAX(SAL) FROM EMP)
```

-----  
**5000**

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

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

**note :-**

**=> outer query can be INSERT/UPDATE/DELETE/SELECT but inner query must be always SELECT.**

**=> delete employee having max experience ?**

```
DELETE
FROM EMP
WHERE HIREDATE = (SELECT MIN(HIREDATE) FROM EMP)
```

**=> transfer employees from NEW YORK loc to CHICAGO loc ?**

<b>EMP</b>		<b>DEPT</b>		
<b>EMPNO</b>	<b>ENAME</b>	<b>DEPTNO</b>	<b>DEPTNO</b>	<b>DNAME LOC</b>
<b>1</b>		<b>10</b>	<b>10</b>	<b>NEW YORK</b>
<b>2</b>		<b>20</b>	<b>20</b>	<b>DALLAS</b>
<b>3</b>		<b>30</b>	<b>30</b>	<b>CHICAGO</b>

```
UPDATE EMP
SET DEPTNO = (SELECT DEPTNO FROM DEPT WHERE LOC='CHICAGO')
WHERE DEPTNO = (SELECT DEPTNO FROM DEPT WHERE LOC='NEW YORK')
```

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

<b>before swap</b>		<b>after swap</b>	
<b>7499</b>	<b>1600</b>	<b>7499</b>	<b>1300</b>
<b>7521</b>	<b>1300</b>	<b>7521</b>	<b>1600</b>

```
UPDATE EMP
SET SAL = CASE EMPNO
            WHEN 7499 THEN (SELECT SAL FROM EMP WHERE EMPNO=7521)
            WHEN 7521 THEN (SELECT SAL FROM EMP WHERE EMPNO=7499)
            END
WHERE EMPNO IN (7499,7521)
```

## Multi-row subqueries :-

-----

=> if inner query 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

single	multi
=	IN
<>	NOT IN
>	>ANY >ALL
<	<ANY <ALL

03-aug-23

=> for which dept employee smith,blake working ? display dept name ?

```
select dname
from dept
where deptno IN (select deptno
                 from emp
                 where ename IN ('SMITH','BLAKE'))
```

=> display employee name & dept name of smith ,blake ?

```
select e.ename,d.dname
from emp as e inner join dept as d
on e.deptno = d.deptno
where e.ename IN ('SMITH','BLAKE')
```

## SUBQUERY VS JOIN :-

-----

1 to display data from one table and condition based on another table then use subquery or join

2 to display data from two tables then use join

## ANY operator :-

-----

=> use ANY for > < comparision with multiple values

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

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

**4500 TRUE**

**WHERE X < ANY(1000,2000,3000)**

<b>IF X = 800</b>	<b>TRUE</b>
<b>1500</b>	<b>TRUE</b>
<b>4500</b>	<b>FALSE</b>

**ALL :-**

-----

**=> use ALL for > < comparision with multiple**

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

<b>IF X = 800</b>	<b>FALSE</b>
<b>1500</b>	<b>FALSE</b>
<b>4500</b>	<b>TRUE</b>

**WHERE X < ALL(1000,2000,3000)**

<b>IF X=800</b>	<b>TRUE</b>
<b>1500</b>	<b>FALSE</b>
<b>4500</b>	<b>FALSE</b>

**=> employees earning more than all managers ?**

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

**co-related subqueries :-**

-----

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

**=> execution starts from outer query and inner query is executed no of times depends no of rows in a table.**

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

**example 1 :-**

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

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

```

SELECT *
FROM EMP AS X
WHERE SAL > (SELECT AVG(SAL)
             FROM EMP
             WHERE DEPTNO = X.DEPTNO)

```

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

=> find employees earning max sal in their dept ?

```

SELECT *
FROM EMP AS X
WHERE SAL = (SELECT MAX(SAL)
             FROM EMP
             WHERE DEPTNO = X.DEPTNO)

```

=> display top 3 max salaries ?

SAL  
 5000  
 1000  
 3000  
 2000  
 4000

```

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

```

EMP A SAL	EMP B SAL		
5000	5000	3 > (0)	TRUE
1000	1000	3 > (4)	FALSE
3000	3000	3 > (2)	TRUE
2000	2000	3 > (3)	FALSE
4000	4000	3 > (1)	TRUE

04-aug-23

=> display 5th max salary ?

```

SELECT DISTINCT A.SAL
FROM EMP AS A
WHERE (5-1) = (SELECT COUNT(DISTINCT B.SAL)
              FROM EMP AS B
              WHERE A.SAL < B.SAL)

```

**WHERE A.SAL < B.SAL)**

## **DERIVED TABLES :-**

-----

**=> subqueries in FROM clause are called derived tables**

```
SELECT columns  
FROM (SELECT STATEMENT) AS <ALIAS>  
WHERE COND
```

**=> subquery output acts like a table for outer query**

**=> derived tables are used in following scenarios**

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

## **controlling order of execution :-**

-----

### **default order :-**

-----

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

**=> use derived table to control this order of execution**

### **example 1 :-**

**=> display employees ranks based on sal ?**

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

**above query displays ranks of all the employees but to display top 3 employees**

```
SELECT EMPNO,ENAME,SAL,  
       DENSE_RANK() OVER (ORDER BY SAL DESC) AS RNK  
FROM EMP  
WHERE RNK<=3 => ERROR
```

**column alias cannot be used in where clause because where clause is executed before select , to overcome this use derived table.**

```
SELECT *
```

```

FROM (SELECT EMPNO,ENAME,SAL,
            DENSE_RANK() OVER (ORDER BY SAL DESC) AS RNK
      FROM EMP) AS E
WHERE RNK<=3

```

```

SELECT *
FROM E
WHERE RNK<=3

```

=> display top 5 max salaries ?

```

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

```

=> display 5th max salary ?

```

WHERE RNK = 5

```

Example 2 :-

=> display first 5 rows from emp table ?

```

SELECT *
FROM (SELECT ROW_NUMBER() OVER (ORDER BY EMPNO ASC) AS RNO,
            EMPNO,ENAME,SAL
      FROM EMP) AS E
WHERE RNO <= 5

```

```

WHERE RNO IN (5,7,10)

```

```

WHERE RNO BETWEEN 5 AND 10

```

```

WHERE RNO%2=0

```

=> display last 3 rows from emp table ?

```

SELECT *
FROM (SELECT ROW_NUMBER() OVER (ORDER BY EMPNO ASC) AS RNO,
            EMPNO,ENAME,SAL
      FROM EMP) AS E
WHERE RNO >= (SELECT COUNT(*)-2 FROM EMP)

```

=> delete first 3 rows ?

```

DELETE
FROM (SELECT ROW_NUMBER() OVER (ORDER BY EMPNO ASC) AS RNO,
            EMPNO,ENAME,SAL
      FROM EMP) AS E
WHERE RNO <= 3 => ERROR

```

**NOTE :-** in derived table outer query cannot be DML command and is must be always **SELECT**

**CTE :-**

-----

=> **CTE** stands for common table expression.

=> using CTE we can give name to the query output and we can reference that name in another query like **SELECT/INSERT/UPDATE/DELETE**.

=> using CTE we can simplify complex operations

syn :-

**WITH <CTE-NAME>**

**AS**

**(SELECT STATEMENT)**

**SELECT / INSERT / UPDATE / DELETE**

**Ex :-**

=> delete first 5 rows ?

**WITH E**

**AS**

**(SELECT ROW\_NUMBER() OVER (ORDER BY EMPNO ASC) AS RNO,  
EMPNO,ENAME,SAL**

**FROM EMP)**

**DELETE FROM E WHERE RNO<=5**

**05-AUG-23**

=> delete duplicate rows ?

**EMP77**

**ENO ENAME SAL**

**1 A 5000**

**2 B 6000**

**3 C 7000**

**1 A 5000**

**2 B 6000**

**STEP 1 :-**

**SELECT ENO,ENAME,SAL,**

**ROW\_NUMBER() OVER (PARTITION BY ENO,ENAME,SAL  
ORDER BY ENO ASC) AS RNO**

**FROM EMP77**

**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 with rno > 1**

**WITH E**

**AS**

```
( SELECT ENO,ENAME,SAL,
      ROW_NUMBER() OVER (PARTITION BY ENO,ENAME,SAL
                        ORDER BY ENO ASC) AS RNO
  FROM EMP77)
```

**DELETE FROM E WHERE RNO > 1**

**scalar subqueries :-**

-----

**=> subqueries in SELECT are called scalar subqueries**

```
SELECT (subquery1),(subquery2),-----
FROM tablename
WHERE cond
```

**=> 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
9	4

**example 2 :-**

**=> display dept wise total salary ?**

```
select deptno,sum(sal) as dept_totsal
from emp
group by deptno
```

10	8750.00
20	7100.00
30	5300.00

**=> display deptno dept\_totsal totals ?**

```
select deptno,sum(sal) as dept_totsal ,
      (select sum(sal) from emp) as totals
from emp
group by deptno
```

10	8750.00	21150.00
----	---------	----------



20 7100.00 21150.00  
30 5300.00 21150.00

=> display deptno dept\_totsal totalsal pct ?

$$pct = ( dept\_totalsal / totalsal ) * 100$$

```
select deptno,sum(sal) as dept_totsal ,  
       (select sum(sal) from emp) as totalsal ,  
       (sum(sal)/(select sum(sal) from emp) ) * 100 as pct  
from emp  
group by deptno
```

=====

05-aug-23

### DB SECURITY :-

-----

- 1 LOGINS           => provides security at server level
- 2 USERS           => provides security at db level
- 3 PRIVILEGES      => provides security at table level
- 4 VIEWS           => provides security at row & col level

server (login)  
  database (user)  
    table (privileges)  
      rows & cols (views)

creating logins :-

-----

=> in object explorer

select security => logins => New login

- 1 Enter Login Name :- NARESH
- 2 select SQL SERVER Authentication
- 3 enter password :- 123  
  
confirm password :- 123
- 4 uncheck user must change password at next login

click OK

command to create new login :-

-----

USE [master]  
GO

**CREATE LOGIN [NARESH] WITH PASSWORD='123'**  
**GO**

**NOTE :-** using this login NARESH can connect to server but cannot access database, to access database NARESH must be associated with a user in database.

**creating user in database :-**  
-----

**=> open the db in which you want to create user**

**DB6PM**  
**SECURITY**  
**USERS => NEW USER**

**=> Enter username :- VIJAY**  
**Enter loginname :- NARESH**

**=> click ok**

**=> login NARESH associated with user VIJAY in db DB6PM**

**command to create user :-**  
-----

**USE [BATCH12]**  
**GO**  
**CREATE USER [VIJAY] FOR LOGIN [NARESH]**  
**GO**

**07-AUG-23**

**SERVER**  
**SA**  
**NARESH**

**BATCH12**  
**DBO (SA)**  
**EMP**  
**DEPT**  
**CUST**  
**STUDENT**  
**VIJAY (NARESH)**

**Granting permissions (privileges) :-**  
-----

**=> "GRANT" command is used to grant permissions from one user to another user.**

**syn :- GRANT <permissions> ON <tablename> TO <username>**

**DBO :-**

-----

**GRANT SELECT,INSERT,UPDATE,DELETE ON EMP TO VIJAY**

**VIJAY :-**

-----

- 1 SELECT \* FROM EMP**
- 2 UPDATE EMP SET SAL = 3000 WHERE EMPNO = 7698**
- 3 DELETE FROM EMP WHERE EMPNO = 7698**

**NOTE :-** changes made by "vijay" visible to "dbo"

**REVOKE command :-**

-----

**=> command used to take back permissions from user**

**syn :- REVOKE <permissions> ON <tablename> FROM <username>**

**DBO :-**

-----

**REVOKE SELECT,INSERT,UPDATE,DELETE ON EMP FROM VIJAY**

**VIJAY :-**

-----

**SELECT \* FROM EMP => ERROR**

**DB Objects / SCHEMA objects :-**

-----

**TABLES**

**VIEWS**

**SYNONYMS**

**SEQUENCES**

**INDEXES**

**VIEWS :-**

-----

**=> a view is a subset of a table i.e. part of the table.**

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

**=> a view represents a query**

**=> views are created**

- 1 to provide security**
- 2 to reduce complexity**

**=> with the help of views we can provide another level of security called row & column level i.e. using view we can grant specific rows and columns to user.**

**=> views are 2 types**

- 1 simple views**
- 2 complex views**

**simple views :-**  
-----

**=> a view said to be simple view if it is created on single table.**

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

**EX :-**

```
CREATE VIEW V1  
AS  
SELECT EMPNO,ENAME,JOB,DEPTNO FROM EMP
```

**=> sql server creates view "v1" and stores query but not query output (data)**

```
SELECT * FROM V1
```

**=> sql server executes the above query as follows**

```
SELECT * FROM (SELECT EMPNO,ENAME,JOB,DEPTNO FROM EMP)
```

**Granting permissions on view to user :-**  
-----

**DBO :-**  
-----

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

**VIJAY :-**  
-----

```
1 SELECT * FROM V1
```

```
2 UPDATE V1 SET JOB='ANALYST' WHERE EMPNO = 7698
```

**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 :-**  
-----

**SELECT \* FROM V2**

**complex views :-**  
-----

**=> a view said to be complex view**

- 1 if based on multiple tables**
- 2 if query contains group by**
  - distinct**
  - aggregate functions**
  - set operators**
  - subqueries**

**=> 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 AS E INNER JOIN DEPT AS D  
      ON E.DEPTNO = D.DEPTNO**

**=> after creating view whenever we want data from emp & dept tables  
     then instead of writing join query write the simple query**

**SELECT \* FROM CV1**

**example 2 :-**

**CREATE VIEW CV2  
AS  
SELECT DEPTNO,MIN(SAL) AS MINSAL,  
          MAX(SAL) AS MAXSAL,  
          SUM(SAL) AS TOTSAL,  
          COUNT(\*) AS CNT  
FROM EMP  
GROUP BY DEPTNO**

**=> after creating whenever we want dept wise summary then execute the  
     following query**

**SELECT \* FROM CV2**

**=> difference between simple and complex views ?**

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

**=> list of views ?**

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

**=> list of tables created by user ?**

```
SELECT *  
FROM INFORMATION_SCHEMA.TABLES  
WHERE TABLE_TYPE='BASE TABLE'
```

**Dropping views :-**  
-----

```
DROP VIEW V1
```

**08-AUG-23**

**synonyms :-**  
-----

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

**=> if tablename is lengthy we can give a simple or short name to the table called synonym and instead of using tablename we can use synonym name in select/insert/update/delete queries.**

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

```
ex :- CREATE SYNONYM E FOR EMP
```

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

```
1 SELECT * FROM E
```

```
2 UPDATE E SET COMM=500 WHERE EMPNO = 7844
```

**accessing tables without db & schema name :-**  
-----

```
SELECT * FROM DB2PM.DBO.CUST
```

**CREATE SYNONYM CUST FOR DB2PM.DBO.CUST**

**SELECT \* FROM CUST**

**Question :-**

**1 CREATE SYNONYM E FOR EMP**

**2 SELECT \* FROM EMP AS E**

**3 SP\_RENAME 'EMP','E' => changes tablename from emp to e**

**difference between synonym and alias ?**

	<b>synonym</b>	<b>alias</b>
<b>1</b>	<b>permanent</b>	<b>not permanent</b>
<b>2</b>	<b>stored in db</b>	<b>not stored in db</b>
<b>3</b>	<b>scope of the synonym is upto the schema</b>	<b>scope of the alias is upto the query</b>

**=> list of synonyms created ?**

**SELECT \* FROM SYS.SYNONYMS**

**Dropping synonym :-**

-----

**DROP SYNONYM E**

**SEQUENCE :-**

-----

**=> sequence is created to generate sequence numbers for primary key columns.**

**=> used to auto increment column values.**

**ex :-**

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

**using sequence :-**

-----

**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 SUTDENT

```

SID	SNAME
1	A
2	B
3	C
4	D
5	E

example 2 :-

```

CREATE SEQUENCE S2
START WITH 100
INCREMENT BY 1
MAXVALUE 9999

```

=> use above sequence to update empno ?

```

UPDATE EMP SET EMPNO = NEXT VALUE FOR S2

```

example 3 :-

BILLNO	BDATE	AMOUNT
NIT/0823/1	??	1000
NIT/0823/2	??	??

```

CREATE TABLE BILL
(
BILLNO VARCHAR(20) ,
BDATE DATETIME,
AMOUNT MONEY
)

```

```

CREATE SEQUENCE S5
START WITH 1
INCREMENT BY 1
MAXVALUE 9999

```

=> use above sequence to generate billno ?

```

INSERT INTO BILL
VALUES('NIT/' +
      FORMAT(GETDATE(), 'MMyy') + '/' +
      CAST(NEXT VALUE FOR S5 AS VARCHAR) , GETDATE(), 1000)

```



**SELECT \* FROM BILL**

<b>NIT/0823/1</b>	<b>2023-08-08 12:17:21.670</b>	<b>1000.00</b>
<b>NIT/0823/2</b>	<b>2023-08-08 12:17:25.000</b>	<b>1000.00</b>

**how to restart sequence ?**

**ALTER SEQUENCE S1 RESTART WITH 1**

**=> list of sequences created by user ?**

**SELECT \* FROM INFORMATION\_SCHEMA.SEQUENCES**

**Dropping sequence :-**

-----

**DROP SEQUENCE S1**

**IDENTITY :-**

-----

**=> IDENTITY is also used to generate sequence numbers.**

**=> used to auto increment column values.**

**IDENTITY(SEED,INCR)**

**ex :-**

```
CREATE TABLE CUST  
(  
  CID INT IDENTITY (100,1) ,  
  CNAME VARCHAR(10)  
)
```

```
INSERT INTO CUST(CNAME) VALUES('A')  
INSERT INTO CUST(CNAME) VALUES('B')  
INSERT INTO CUST(CNAME) VALUES('C')  
INSERT INTO CUST(CNAME) VALUES('D')
```

**SELECT \* FROM CUST**

<b>CID</b>	<b>CNAME</b>
<b>100</b>	<b>A</b>
<b>101</b>	<b>B</b>
<b>102</b>	<b>C</b>
<b>103</b>	<b>D</b>

**how to reset identity ?**

**DBCC CHECKIDENT('CUST',RESEED,99)**

**DBCC => DB CONSISTENCY CHECK**

**=> difference between identity & sequence ?**

## **identity**

- 1 always bind to a column in a table**
- 2 value of identity cannot be accessed**
- 3 cannot be declared with maxvalue**

## **sequence**

- not bind a any column**
- value of sequence can be accessed by using next value for sequence**
- can be declared with maxvalue**

**09-aug-23**

## **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 , in db using index a particular record can be located fastly.**

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

**=> indexes created on columns**

- 1 which are frequently used in where clause**
- 2 which are used in join operation**

## **Types of indexes :-**

-----

- 1 Non Clustered**
- 2 Clustered**

## **Non Clustered :-**

-----

**CREATE INDEX <NAME> ON <TABNAME> (COLNAME)**

**EX :- CREATE INDEX I1 ON EMP (SAL)**

**=> after executing above command sql server creates a structure called btree i..e balance binary tree.**

<b>EMP</b>		<b>3000</b>		
<b>SAL</b>				
<b>1000</b>				
<b>4000</b>		<b>2000</b>		<b>4000</b>
<b>3000</b>				
<b>5000</b>	<b>1000 *</b>	<b>2500 *</b>	<b>4000 *</b>	<b>5000 *</b>
<b>1500</b>	<b>1500 *</b>	<b>3000 *,*</b>		
<b>3000</b>	<b>2000 *</b>			

2500  
2000

=> when user submits the query sql server uses following methods to locate the record

- 1 TABLE SCAN
- 2 INDEX SCAN

=> in table scan , sql server scans complete table

=> in index scan , sq server scans half of the table , so index scan is much faster than table scan.

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

```
select * from emp (table scan)
select * from emp where ename='blake' (table scan)
```

unique index :-  
-----

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

ex :- CREATE UNIQUE INDEX I2 ON EMP(ENAME)

K			
G		Q	
ADAMS *	JAMES *		
ALLEN *		MARTIN *	SCOTT *
BLAKE *		MILLER *	SMITH *

- 1 SELECT \* FROM EMP WHERE ENAME='BLAKE'
- 2 INSERT INTO EMP(EMPNO,ENAME,SAL)  
VALUES(100,'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 automatically indexed by sql server and sql server creates unique index on pk / unique columns and unique index doesn't allow duplicates so pk / unique also doesn't allow duplicates.

CLUSTERED INDEX :-  
-----

=> a non clustered index stores addresses of the actual records stored in table where as clustered index stores actual records.

ex :-

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

```
CREATE CLUSTERED INDEX I10 ON CUST(CID)
```

```
INSERT INTO cust VALUES(10,'A')
INSERT INTO cust VALUES(80,'B')
INSERT INTO cust VALUES(40,'C')
INSERT INTO cust VALUES(60,'D')
```

50

30                      70

10   A            40   C            60   D   80   B

**SELECT \* FROM CUST => sql server goes to cluster index and access all leaf nodes from left to right**

10 A  
40 C  
60 D  
80 B

**NOTE :-**

=> only one clustered index is allowed per table

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

=> difference between non clustered and clustered indexes ?

non clustered	clustered
1 stores addr of actual record	stores actual record
2 needs extra storage	doesn't need extra storage
3 requires two lookups to access the records	requires one lookup to access the record
4 sql server allows 999 non clustered indexes per table	allows only one clustered index per table

**5 created on unique column**

**created on primary key column**

**=> list of indexes ?**

**sp\_helpindex emp**

**dropping index :-**

-----

**DROP INDEX EMP.I1**

**SERVER**

**DATABASE**

**TABLES**

**ROWS & COLS**

**CONSTRAINTS**

**INDEXES**

**TRIGGERS**

**VIEWS**

**SYNONYMS**

**SEQUENCES**

**CREATING NEW TABLE FROM EXISTING TABLE :- (replica)**

-----

**SELECT columns INTO <new-tabname>  
FROM <old-tabname>  
[WHERE cond]**

**example 1 :- (copying complete table)**

**SELECT \* INTO EMP10  
FROM EMP**

**example 2 :- (copying specific rows & cols)**

**SELECT EMPNO,ENAME,JOB,SAL INTO EMP11  
FROM EMP  
WHERE JOB IN ('CLERK','MANAGER')**

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

**SELECT \* INTO EMP12  
FROM EMP  
WHERE 1=2**

**example 4 :- (copy table from one db to another db)**

**SELECT \* INTO DB2PM.DBO.ACCOUNTS  
FROM DB6PM.DBO.ACCOUNTS**

**above command copies accounts table from db6pm db to db2pm db**

**copying data from one table to another table :-**  
-----

**INSERT INTO <TARGET-TABLE>  
SELECT COLUMNS FROM <SOURCE-TABLE> [WHERE COND]**

**ex :-**

**copy data from emp to emp12 ?**

**INSERT INTO EMP12  
SELECT \* FROM EMP**

**11-aug-23**

**MERGE command :-**  
-----

- => command used to merge data into a table.**
- => merge is the combination of insert,update and delete.**
- => used to manage replicas.**
- => using merge command we can apply changes made to source table to replica.**

**syn :-**

**MERGE INTO <TARGET-TABLE> AS <ALIAS>  
USING <SOURCE-TABLE> AS <ALIAS>  
ON (CONDITION)  
WHEN MATCHED THEN  
    UPDATE  
WHEN NOT MATCHED THEN  
    INSERT  
WHEN NOT MATCHED BY SOURCE THEN  
    DELETE ;**

**example :-**

**step 1 :- create source table**

**create table custs  
(  
    cid int ,  
    cname varchar(10),  
    addr varchar(10)  
)**

**insert into custs values(10,'A','HYD'),(11,'B','MUM')**

**step 2 :- create replica**

**select \* into custt from custs**

**step 3 :- modify the source table**

**1 insert into custs values(12,'C','DEL')**

**2 update custs set addr='BLR' where cid = 10**

#### **CUSTS**

<b>CID</b>	<b>CNAME</b>	<b>ADDR</b>
<b>10</b>	<b>A</b>	<b>BLR =&gt; UPDATED</b>
<b>11</b>	<b>B</b>	<b>MUM</b>
<b>12</b>	<b>C</b>	<b>DEL =&gt; INSERTED</b>

**step 4 :- apply changes to replica**

```
MERGE INTO CUST AS T
USING CUSTS AS S
ON (S.CID = T.CID)
WHEN MATCHED THEN
    UPDATE SET T.ADDR = S.ADDR
WHEN NOT MATCHED THEN
    INSERT VALUES(S.CID,S.CNAME,S.ADDR)
WHEN NOT MATCHED BY SOURCE THEN
    DELETE ;
```

#### **SQL**

<b>COMMANDS</b>	<b>OPERATIONS</b>	<b>FUNCTIONS</b>	<b>OBJECTS</b>
<b>DDL</b>	<b>WHERE</b>	<b>DATE</b>	<b>TABLES</b>
<b>DML</b>	<b>ORDER BY</b>	<b>STRING</b>	<b>VIEWS</b>
<b>DQL</b>	<b>DISTINCT</b>	<b>NUMERIC</b>	<b>SYNONYMS</b>
<b>TCL</b>	<b>TOP</b>	<b>CONVERSION</b>	<b>SEQUENCES</b>
<b>DCL</b>	<b>GROUP BY</b>	<b>SPECIAL</b>	<b>INDEXES</b>
	<b>JOINS</b>	<b>ANALYTICAL</b>	
	<b>SET OPERATIONS</b>	<b>AGGREGATE</b>	
	<b>SUBQUERIES</b>		

=====

#### **T-SQL programming (Transact-SQL)**

-----

**introduction to t-sql programming**

**conditional stmts**

**loops**

**error handling**

**cursos**

**procedures**

**functions**

**triggers**

#### **SQL**

#### **T-SQL**

**1 submit one by one command**

**submit group of commands**

**2 doesn't support conditional**

**supports conditional stmts**

**statements**

- |   |                                |
|---|--------------------------------|
| <b>3 doesn't support loops</b>          | <b>T-SQL supports loops</b>    |
| <b>4 doesn't support error handling</b> | <b>supports error handling</b> |
| <b>5 doesn't support reusability</b>    | <b>supports reusability</b>    |

**=> T-SQL programs are called T-SQL blocks**

**=> T-SQL blocks are 2 types**

- 1 Anonymous Blocks**
- 2 Named Blocks**
  - procedures**
  - functions**
  - triggers**

**Anonymous Blocks :-**

-----

**=> the following statements are used in anonymous blocks**

- 1 declare**
- 2 set**
- 3 print**

**Declare stmt :-**

-----

**=> used to declare variables**

**DECLARE @varname datatype(size)**

**ex :-**

**DECLARE @X INT  
DECLARE @S VARCHAR(10)  
DECLARE @D DATE**

**DECLARE @X INT,@S VARCHAR(10),@D DATE**

**SET stmt :-**

-----

**=> used to assign value to variable**

**SET @var = value**

**ex :-**

**SET @X = 100  
SET @S = 'ABC'  
SET @D = GETDATE()**

**PRINT stmt :-**



-----  
**PRINT 'hello'**  
**PRINT @X**

**example 1 :-**

**DECLARE @X INT,@Y INT,@Z INT**  
**SET @X=100**  
**SET @Y=200**  
**SET @Z = @X + @Y**  
**PRINT @Z**

**example 2 :-**

**DECLARE @D DATE**  
**SET @D = '2023-08-15'**  
**PRINT DATENAME(DW,@D)**

**example 3 :-**

**DECLARE @MNAME VARCHAR(20),@LNAME VARCHAR(20)**  
**SET @S = 'SACHIN TENDULKAR'**  
**SET @FNAME = SUBSTRING(@S,1,CHARINDEX(' ',@S)-1)**  
**SET @LNAME = SUBSTRING(@S,CHARINDEX(' ',@S)+1,LEN(@S))**  
**PRINT @FNAME**  
**PRINT @LNAME**

**DB programming :-**  
-----

**=> to perform operations on db execute sql commands from t-sql program**  
**=> the following commands can be executed from t-sql program.**

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

**12-AUG-23**

**SELECT syntax :-**  
-----

**SELECT @VAR1 = COL1 , @VAR2 = COL2 , -----**  
**FROM TABNAME**  
**WHERE COND**

**ex :-**

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

**DECLARE @ENO INT ,@NAME VARCHAR(10),@SAL MONEY**  
**SET @ENO = 107**  
**SELECT @NAME = ENAME,@SAL = SAL**

```
FROM EMP
WHERE EMPNO = @ENO
PRINT @NAME + ' ' + CAST(@SAL AS VARCHAR)
```

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

```
DECLARE @ENO INT ,@HIRE DATE ,@EXPR TINYINT
SET @ENO = 100
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 sal by specific amount  
after increment if sal exceeds 5000 then cancel that increment ?

```

DECLARE @ENO INT,@AMT MONEY,@SAL MONEY
SET @ENO = 102
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 process bank transaction (w/d) ?

```

ACCOUNTS
ACCNO      ACTYPE      BAL
100  S      10000
101  C      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 TYPE'**

**=> write a prog for money transfer ?**

```
DECLARE @SACNO INT,@TACNO INT,@AMT MONEY,@BAL MONEY
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
        UPDATE ACCOUNTS SET BAL = BAL -@AMT WHERE ACCNO = @SACNO
        UPDATE ACCOUNTS SET BAL = BAL+ @AMT WHERE ACCNO = @TACNO
    END
```

**14-AUG-23**

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

**STUDENT**

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

**RESULT**

SNO	TOTAL	AVG	RESULT
-----	-------	-----	--------

```
DECLARE @SNO INT,@S1 INT,@S2 INT,@S3 INT
DECLARE @TOTAL INT,@AVG DECIMAL(5,2),@RES CHAR(4)
SET @SNO=100
SELECT @S1=S1 , @S2 = S2 , @S3 = S3 FROM STUDENT WHERE SNO=@SNO
SET @TOTAL = @S1 + @S2 + @S3
SET @AVG = @TOTAL/3
IF @S1>=35 AND @S2>=35 AND @S3>=35
    SET @RES='PASS'
ELSE
    SET @RES='FAIL'
INSERT INTO RESULT VALUES(@SNO,@TOTAL,@AVG,@RES)
```

**WHILE LOOP :-**

-----

**WHILE(cond)**

**BEGIN**

statements

**END**

if cond = true loop continues

if cond = false loop terminates

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

```

DECLARE @X INT = 1
WHILE(@X<=20)
BEGIN
    PRINT @X
    SET @X = @X + 1
END

```

=> write a prog to print 2024 calendar ?

```

2024-01-01    ?
2024-01-02    ?

```

```

2024-12-31    ?

```

```

DECLARE @D1 DATE,@D2 DATE
SET @D1 = '2024-01-01'
SET @D2 = '2024-12-31'
WHILE(@D1<=@D2)
BEGIN
    PRINT CAST(@D1 AS VARCHAR) + ' ' + DATENAME(DW,@D1)
    SET @D1 = DATEADD(DD,1,@D1)
END

```

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

```

DECLARE @D1 DATE,@D2 DATE
SET @D1 = '2024-01-01'
SET @D2 = '2024-12-31'
WHILE(@D1<=@D2)
BEGIN
    IF DATENAME(DW,@D1) = 'SUNDAY'
        PRINT CAST(@D1 AS VARCHAR) + ' ' + DATENAME(DW,@D1)
    SET @D1 = DATEADD(DD,1,@D1)
END

```

```

DECLARE @D1 DATE,@D2 DATE
SET @D1 = '2024-01-01'
SET @D2 = '2024-12-31'
/* finding first sunday */
WHILE(DATENAME(DW,@D1)<>'SUNDAY')
BEGIN
    SET @D1 = DATEADD(DD,1,@D1)
END
/* printing sundays */
WHILE(@D1<=@D2)
BEGIN
    PRINT CAST(@D1 AS VARCHAR) + ' ' + DATENAME(DW,@D1)
    SET @D1 = DATEADD(DD,7,@D1)
END

```

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

```

input :-    NARESH

```

**output :-**

**N  
A  
R  
E  
S  
H**

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

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

**input :- NARESH**

**output :-**

**N  
NA  
NAR  
NARE  
NARES  
NARESH**

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

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

**INPUT :- NARESH**

**OUTPUT :- HSERAN**

```
DECLARE @S VARCHAR(10),@X INT ,@R VARCHAR(10)=''  
SET @S = 'NITIN'  
SET @X = LEN(@S)  
WHILE(@X>=1)  
BEGIN  
    SET @R = @R + SUBSTRING(@S,@X,1)  
    SET @X = @X - 1  
END  
PRINT @R  
IF @S = LTRIM(@R)
```

```
PRINT 'palindrome'  
ELSE  
PRINT 'not a palindrome'
```

**16-AUG-23**

**CURSOR :-**  
-----

**=> cursors are used to access row-by-row in t-sql program.**

**=> cursors are used to process multiple rows in t-sql program.**

**=> from t-sql prog if we submit a query , sql server executes the query and data returned by query is copied to temporary memory called cursor and in prog we can give name to the cursor and access row-by-row from the cursor and process the row.**

**=> follow below steps to use cursor**

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

**Declaring cursor :-**  
-----

**DECLARE <NAME> CURSOR FOR SELECT STATEMENT**

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

**Opening cursor :-**  
-----

**OPEN <NAME>**

**EX :- OPEN C1**

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

**Fetching records from cursor :-**  
-----

**=> "FETCH" stmt is used to fetch record from cursor.**

**FETCH NEXT FROM <CURSOR> INTO VARIABLES**

**EX :- FETCH NEXT FROM C1 INTO x,y**

**=> a fetch stmt fetches one row at a time but to process multiple rows fetch stmt**

should be executed multiple times , so fetch stmt should be in a loop.

**closing cursor :-**

-----

**CLOSE <cursor-name>**

**EX :- CLOSE C1**

**deallocate cursor :-**

-----

**DEALLOCATE <cursor-name>**

**EX :- DEALLOCATE C1**

**@@fetch\_status :-**

-----

**=> it is a system variable that returns status of fetch statement**

**0 => if fetch successful**

**-1 => if fetch unsuccessful**

**example 1 :-**

**=> write a prog to print all employee names and salaries ?**

```
DECLARE C1 CURSOR FOR SELECT ENAME,SAL FROM EMP  
DECLARE @NAME VARCHAR(10),@SAL MONEY  
OPEN C1  
FETCH NEXT FROM C1 INTO @NAME,@SAL  
WHILE(@@FETCH_STATUS=0)  
BEGIN  
    PRINT @NAME + ' ' + CAST(@SAL AS VARCHAR)  
    FETCH NEXT FROM C1 INTO @NAME,@SAL  
END  
    CLOSE C1  
    DEALLOCATE C1
```

**=> write a prog to print to calculate total sal without using sum function ?**

```
DECLARE C1 CURSOR FOR SELECT SAL FROM EMP  
DECLARE @SAL MONEY,@TOTSAL MONEY  
OPEN C1  
FETCH NEXT FROM C1 INTO @SAL  
WHILE(@@FETCH_STATUS=0)  
BEGIN  
    SET @TOTSAL = @TOTSAL + @SAL  
    FETCH NEXT FROM C1 INTO @SAL  
END
```



```
PRINT @TOTSAL
CLOSE C1
DEALLOCATE C1
```

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

```
DECLARE C1 CURSOR FOR SELECT SAL FROM EMP
DECLARE @SAL MONEY,@MAX MONEY=0
OPEN C1
FETCH NEXT FROM C1 INTO @SAL
WHILE(@@FETCH_STATUS=0)
BEGIN
    IF @SAL > @MAX
        SET @MAX = @SAL
    FETCH NEXT FROM C1 INTO @SAL
END
PRINT @MAX
CLOSE C1
DEALLOCATE C1
```

```
DECLARE C1 CURSOR FOR SELECT SAL FROM EMP ORDER BY SAL DESC
DECLARE @SAL MONEY
OPEN C1
FETCH NEXT FROM C1 INTO @SAL
PRINT @SAL
CLOSE C1
DEALLOCATE C1
```

=> write a prog to find min sal ?

```
DECLARE C1 CURSOR FOR SELECT SAL FROM EMP ORDER BY SAL ASC
DECLARE @SAL MONEY
OPEN C1
FETCH NEXT FROM C1 INTO @SAL
PRINT @SAL
CLOSE C1
DEALLOCATE C1
```

17-AUG-23

=> 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	60	50

RESULT			
SNO	TOTAL	AVG	RESULT

```
DECLARE C1 CURSOR FOR SELECT SNO,S1,S2,S3 FROM STUDENT
DECLARE @SNO INT,@S1 INT,@S2 INT,@S3 INT
DECLARE @TOTAL INT,@AVG DECIMAL(5,2),@RES CHAR(4)
```

```

OPEN C1
FETCH NEXT FROM C1 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

```

### **SCROLLABLE CURSOR :-**

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

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

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

<b>FETCH FIRST</b>	=> fetches first record
<b>FETCH NEXT</b>	=> fetches next record
<b>FETCH LAST</b>	=> fetches last record
<b>FETCH PRIOR</b>	=> fetches previous record
<b>FETCH ABSOLUTE N</b>	=> fetches Nth record from first record
<b>FETCH RELATIVE N</b>	=> 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

```

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

```

DECLARE C1 CURSOR SCROLL FOR SELECT ENAME FROM EMP
DECLARE @NAME VARCHAR(10)
OPEN C1
FETCH RELATIVE 5 FROM C1 INTO @NAME
WHILE(@@FETCH_STATUS=0)
BEGIN
    PRINT @NAME
    FETCH RELATIVE 5 FROM C1 INTO @NAME
END
CLOSE C1
DEALLOCATE C1

```

**=> 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

```

=====

## **ERROR HANDLING / EXCEPTION HANDLING :-**

- 1 syntax errors**
- 2 logical errors**
- 3 runtime errors**

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

```

ex :- DECLARE @X TINYINT
SET @X = 1000 => RUNTIME ERROR

```

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

**=> to replace system generated message with our own simple and user friendly message then we need to handle runtime error**

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

```

BEGIN TRY
    statement1
    statement2
    ----- => statements causes exception
    statement N

```

```
END TRY  
BEGIN CATCH  
    statements; => stmts handles exception  
END CATCH
```

=> if any stmt in try block causes exception then control is transferred to catch block and executes the statements in catch block.

example 1 :-

```
DECLARE @A TINYINT,@B TINYINT,@C TINYINT  
BEGIN TRY  
SET @A=10  
SET @B=0  
SET @C = @A/@B  
PRINT @C  
END TRY  
BEGIN CATCH  
    PRINT 'something went wrong----try again'  
END CATCH
```

error handling functions :-

-----

<b>1 ERROR_NUMBER()</b>	<b>=&gt; returns error code</b>
<b>2 ERROR_MESSAGE()</b>	<b>=&gt; returns error message</b>
<b>3 ERROR_SEVERITY()</b>	<b>=&gt; returns error severity level</b>
<b>4 ERROR_STATE()</b>	<b>=&gt; returns error state</b>
<b>5 ERROR_LINE()</b>	<b>=&gt; returns line number</b>

18-AUG-23

Example 1 :-

```
DECLARE @A TINYINT,@B TINYINT,@C TINYINT  
BEGIN TRY  
SET @A=10  
SET @B=0  
SET @C = @A/@B  
PRINT @C  
END TRY  
BEGIN CATCH  
    IF ERROR_NUMBER()=220  
        PRINT 'value exceeding limit'  
    ELSE IF ERROR_NUMBER()=8134  
        PRINT 'divisor cannot be zero'  
END CATCH
```

Example 2 :-

```
CREATE TABLE EMP44  
(  
    EMPNO INT PRIMARY KEY,  
    ENAME VARCHAR(10) NOT NULL,
```

```
SAL    MONEY CHECK(SAL>=3000)
)
```

**write a prog to insert data into emp44 table ?**

```
DECLARE @ENO INT,@NAME VARCHAR(10),@SAL MONEY
BEGIN TRY
SET @ENO=100
SET @NAME='A'
SET @SAL=5000
INSERT INTO EMP44 VALUES(@ENO,@NAME,@SAL)
END TRY
BEGIN CATCH
    IF ERROR_NUMBER()=2627
        PRINT 'empno should not be duplicate'
    ELSE IF ERROR_NUMBER()=515
        PRINT 'name should not be null'
    ELSE IF ERROR_NUMBER()=547
        PRINT 'sal>=3000'
END CATCH
```

**USER DEFINED ERRORS :-**

-----

**=> errors raised by user are called user defined errors**

**=> user can raise error by using RAISERROR procedure**

**RAISERROR(msg,severity level,state)**

**severity level => 0 to 25**

**0 - 10 => informational message**

**11-19 => errors**

**20-25 => fatal errors**

**state => 0 to 255 => if the same error raised in multiple locations  
then using this state we can identity which  
part of the program causing the error**

**example 1 :-**

```
DECLARE @A TINYINT,@B TINYINT,@C TINYINT
BEGIN TRY
SET @A=10
SET @B=1
IF @B=1
    RAISERROR('divisor cannot be one',16,1)
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'
ELSE
    PRINT ERROR_MESSAGE()
END TRY

BEGIN CATCH
    IF ERROR_NUMBER()=220
        SET @MSG = 'value exceeding limit'
    ELSE IF ERROR_NUMBER()=8134
        SET @MSG = 'divisor cannot be zero'
    ELSE
        SET @MSG = ERROR_MESSAGE()
    RAISERROR(@MSG,16,1)
END CATCH

```

=> write a prog to input empno and increment sal by specific amount and sunday updates are not allowed ?

```

DECLARE @ENO INT,@AMT MONEY
SET @ENO = 108
SET @AMT=1000
IF DATENAME(DW,GETDATE())='SUNDAY'
    RAISERROR('sunday not allowed',16,1)
ELSE
    UPDATE EMP SET SAL = SAL + @AMT WHERE EMPNO = @ENO

```

list of errors ?

```
SELECT * FROM SYS.MESSAGES
```

```
SELECT * FROM SYS.MESSAGES WHERE MESSAGE_ID=220
```

How to add user define error to sys.messages ?

sp\_addmessage error number,severity level,error msg

ex :-

```
sp_addmessage 50001,16,'sunday not allowed'
```

19-aug-23

raising error by using code :-

-----

```

DECLARE @ENO INT,@AMT MONEY
SET @ENO = 108
SET @AMT=1000
IF DATENAME(DW,GETDATE())='SUNDAY'
    RAISERROR(50001,16,1)
ELSE
    UPDATE EMP SET SAL = SAL + @AMT WHERE EMPNO = @ENO

```

=> write a prog for money withdrawl ?

#### **ACCOUNTS**

<b>ACCNO</b>	<b>ACTYPE</b>	<b>BAL</b>
100	S	10000

```
DECLARE @ACNO INT,@AMT MONEY,@BAL MONEY ,@CNT INT,@MSG VARCHAR(100)  
BEGIN TRY  
SET @ACNO = 100  
SET @AMT=2000  
SELECT @CNT=COUNT(*) FROM ACCOUNTS WHERE ACCNO = @ACNO  
/* check account is valid or not */  
IF @CNT=0  
    RAISERROR('invalid accno',16,1)  
SELECT @BAL = BAL FROM ACCOUNTS WHERE ACCNO = @ACNO  
/* checking balance */  
IF @AMT > @BAL  
    RAISERROR('insufficient balance',16,1)  
UPDATE ACCOUNTS SET BAL = BAL - @AMT WHERE ACCNO = @ACNO  
END TRY  
BEGIN CATCH  
    SET @MSG = ERROR_MESSAGE()  
    RAISERROR(@MSG,16,1)  
END CATCH
```

how to remove user define error from sys.messages ?

**SP\_DROPMESSAGE 50001**

=====

#### **NAMED T-SQL BLOCKS :-**

-----

- 1 STORED PROCEDURES**
- 2 FUNCTIONS**
- 3 TRIGGERS**

#### **SUB-PROGRAMS :-**

-----

- 1 STORED PROCEDURES**
- 2 FUNCTIONS**

#### **Advantages :-**

-----

- 1 modular programming :-**

-----

=> with the help of proc/func a big t-sql program can be divided into small modules.

- 2 reusability :-**

-----  
=> these programs are created with name and also stored in db , so applications which are connected to db can reuse proc/func.

### **3 security :-**

-----

=> because these programs are stored in db , so only authorized users can execute these programs.

### **4 invoked from front-end :-**

-----

=> these programs can be called from front-end applications like java / .net / python etc.

### **5 improves performance :-**

-----

=> proc/func improves performance because they are precompiled i.e. when we create a procedure program is compiled and stored in db and whenever we call procedure only execution is repeated but not compilation , so this improves performance.

### **STORED PROCEDURES :-**

-----

=> a stored procedure is named T-SQL block that accepts some input performs some action on db and may or may not returns a value.

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

**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 (DEFAULT)**
- 2 OUTPUT**

=> INPUT parameter always receives value

=> OUTPUT parameter always sends value

**Example 1 :- procedure without parameters**



**=> create procedure to increment all the employee salaries by 1000 ?**

```
CREATE OR ALTER PROCEDURE raise_salary  
AS  
    UPDATE EMP SET SAL = SAL + 1000
```

**procedure created ( compiled + stored in db)**

**execution :-**  
-----

```
EXECUTE raise_salary
```

**Example 2 :- procedure with parameters**

**=> 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
```

**21-aug-23**

**Execution :-**

```
EXECUTE raise_salary => ERROR
```

```
EXECUTE raise_salary 7369,1000          (positional association)
```

```
EXECUTE raise_salary @eno=7369,@amt=1000 (named association)
```

```
EXECUTE raise_salary @amt=1000,@eno=7369
```

**Example 3 :- (procedure with output parameters)**

**=> create a procedure to increment specific employee sal by specific amount  
and after increment send the updated sal to calling program ?**

```
CREATE OR ALTER PROCEDURE 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 7369,1000,@S OUTPUT
```

**PRINT @S**

**EXECUTE raise\_salary @ENO=7369,@AMT=1000,@NEWSAL=@S OUTPUT**

**Declaring parameters with default value :-**  
-----

**=> a parameter can be declared with default value as follows**

**@amt MONEY = 500**

**=> while executing procedure if we don't pass value to parameter then sql server assigns default value.**

**example 4 :-**

```
CREATE OR ALTER PROCEDURE raise_salary  
@ENO INT,  
@AMT MONEY = 500  
@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 7369,default,@s output  
print @s
```

```
declare @s money  
execute raise_salary @eno=7369,@newsal=@s output  
print @s
```

**example 5 :-**  
-----

```
CREATE TABLE EMP88  
(  
EMPNO INT PRIMARY KEY,  
ENAME VARCHAR(10) NOT NULL,  
SAL MONEY CHECK(SAL>=3000)  
)
```

**=> create a procedure to insert data into emp88 table ?**

```
CREATE OR ALTER PROCEDURE INSERT_EMP88  
@ENO INT,  
@NAME VARCHAR(10),  
@SAL MONEY
```

**AS**

```
DECLARE @MSG VARCHAR(100)  
BEGIN TRY  
  INSERT INTO EMP88 VALUES(@ENO,@NAME,@SAL)  
END TRY  
BEGIN CATCH  
  IF ERROR_NUMBER()=2627  
    SET @MSG='empno should not be duplicate'  
  ELSE IF ERROR_NUMBER() =515  
    SET @MSG='name should not be null'  
  ELSE IF ERROR_NUMBER()=547  
    SET @MSG='sal >=3000'  
  RAISERROR(@MSG,16,1)  
END CATCH
```

**execution :-**

**1 EXECUTE insert\_emp88 100,'A',5000 => 1 row affected**

**2 EXECUTE insert\_emp88 100,'B',6000 => ERROR**

**=> create procedure to insert data into emp88 table , if any error raises then  
send error message to calling program ?**

```
CREATE OR ALTER PROCEDURE INSERT_EMP88  
@ENO INT,  
@NAME VARCHAR(10),  
@SAL MONEY,  
@MSG VARCHAR(100) OUTPUT  
AS  
  BEGIN TRY  
    INSERT INTO EMP88 VALUES(@ENO,@NAME,@SAL)  
  END TRY  
  BEGIN CATCH  
    IF ERROR_NUMBER()=2627  
      SET @MSG='empno should not be duplicate'  
    ELSE IF ERROR_NUMBER() =515  
      SET @MSG='name should not be null'  
    ELSE IF ERROR_NUMBER()=547  
      SET @MSG='sal >=3000'  
  END CATCH
```

**execution :-**

-----

```
DECLARE @S VARCHAR(100)  
EXECUTE insert_emp88 101,'B',1000,@S OUTPUT  
PRINT @S
```

**Assignment :-**

-----

**ACCOUNTS**

ACCNO	ACTYPE	BAL
100	S	10000
101	C	20000

## TRANSACTIONS

TRID	TTYPE	TDATE	TAMT	ACCNO
------	-------	-------	------	-------

```
CREATE SEQUENCE S10
START WITH 1
INCREMENT BY 1
MAXVALUE 99999
```

- 1 create procedure for money withdrawl ?
- 2 create procedure for money deposit ?
- 3 create procedure for money transfer ?

procedure for money deposit :-

-----

## CREATE OR ALTER PROCEDURE CREDIT

```
@ACNO INT,
@AMT MONEY
AS
    DECLARE @CNT INT,@MSG VARCHAR(100)
    BEGIN TRY
        SELECT @CNT = COUNT(*) FROM ACCOUNTS WHERE ACCNO = @ACNO
        IF @CNT = 0
            RAISERROR('account does not exists',16,1)
        UPDATE ACCOUNTS SET BAL = BAL + @AMT WHERE ACCNO = @ACNO
        INSERT INTO TRANSACTIONS
            VALUES(NEXT VALUE FOR S10,'D',GETDATE(),@AMT,@ACNO)
    END TRY
    BEGIN CATCH
        SET @MSG = ERROR_MESSAGE()
        RAISERROR(@MSG,16,1)
    END CATCH
```

22-aug-23

## USER DEFINE FUNCTIONS :-

-----

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

=> a function is also a named T-SQL program and that accepts some input performs some calculation and must return a value.

=> functions are created for calculations or to fetch value from db.

syn :-

## CREATE OR ALTER

```
FUNCTION <NAME>(parameters) RETURNS <type>
```

**AS**  
**BEGIN**  
    **STATEMENTS**  
    **RETURN <expr>**  
**END**

**example 1 :-**

**=> create function to calculate experience of the employee ?**

```
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
```

**EXECUTION :-**

- 
- 1 sql commands**
  - 2 another t-sql prog**
  - 3 front-end applications**

**executing from sql commands :-**

- 
- 1 SELECT DBO.GETEXPR(7369) => 43**
  - 2 SELECT EMPNO,ENAME,  
        DBO.GETEXPR(EMPNO) AS EXPR FROM EMP**

**TABLE VALUED FUNCTIONS :-**

- 
- => these functions returns records**
  - => return type of these functions must be TABLE**
  - => return expr must be select statement**
  - => table valued functions allows only one stmt and it must be return stmt**

```
CREATE OR ALTER  
    FUNCTION <NAME>(parameters) RETURNS TABLE  
AS  
    RETURN (SELECT STMT)
```

**Ex :-**

- => create a function that accepts deptno and returns list of employees working for that dept ?**

**CREATE OR ALTER**

```

FUNCTION GETEMPLIST(@d INT) RETURNS TABLE
AS
RETURN (SELECT * FROM EMP WHERE DEPTNO = @d)

```

**Execution :-**

-----

=> table valued functions are invoked in FROM clause.

```

SELECT * FROM DBO.GETEMPLIST(20)

```

**Example 2 :-**

=> create a function that returns 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) AS E
        WHERE RNK<= @n)

```

**execution :-**

```

SELECT * FROM DBO.getTopNEmpList(5)

```

**difference between procedures & functions ?**

<b>PROCEDURES</b>	<b>FUNCTIONS</b>
<b>1 may or may not returns a value</b>	<b>must return a value</b>
<b>2 can return multiple values</b>	<b>always returns one value</b>
<b>3 return values using OUTPUT parameter</b>	<b>returns value using return stmt</b>
<b>4 cannot be called from sql commands</b>	<b>can be called from sql commands</b>
<b>5 dml commands are allowed in procedure</b>	<b>dml commands are not allowed in functions</b>
<b>6 created to perform one or more dml operations over tables</b>	<b>created for calculation or to fetch value from db</b>
<b>7 create procedure to update balance</b>	<b>create function to get balance</b>

=> difference between 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 expr is scalar type variable	return expr is select stmt
4 called in select clause	called in from clause

23-aug-23

Assignment :-

-----

ACCOUNTS  
ACCNO ACTYPEBAL

TRANSACTIONS  
TRID TTYPE TDATE TAMT ACCNO

CREATE SEQUENCE S10  
START WITH 1  
INCREMENT BY 1  
MAXVALUE 99999

=> create following procedures & function to implement various bank transactions ?

- 1 account opening (proc)
- 2 account closing (proc)
- 3 balance enquiry (svf)
- 4 money deposit (proc)
- 5 money withdrawl (proc)
- 6 money transfer (proc)
- 7 statement between two given dates (tvf)
- 8 latest N transactions of particular customer (tvf)

list of procedures & functions ?

SELECT \* FROM INFORMATION\_SCHEMA.ROUTINES

DROPING PROCEDURES & FUNCTIONS :-

-----

DROP PROCEDURE RAISE\_SALARY

DROP FUNCTION GETEXPR

TRIGGERS :-

-----  
=> trigger is also a named T-SQL block like procedure but executed implicitly by sql server whenever user submits dml commands.

=> triggers are created

- 1 to control dml operations
- 2 to enforce complex rules and validations
- 3 to audit day-to-day operations on tables

syn :-

```
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 triggers :-  
-----

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

example 1 :-

=> create trigger to not to allow dmls on emp table on sunday ?

```
CREATE OR ALTER TRIGGER T1
ON EMP
AFTER INSERT,UPDATE,DELETE
AS
    IF DATENAME(DW,GETDATE())='SUNDAY'
        BEGIN
            ROLLBACK
            RAISERROR('sunday not allowed',16,1)
        END
```

Testing :-

UPDATE EMP SET SAL=2000 WHERE EMPNO = 7369 => ERROR

=> 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
        BEGIN
            IF DATEPART(HH,GETDATE()) < 10
                OR
                DATEPART(HH,GETDATE()) >= 16
                BEGIN
                    ROLLBACK
                    RAISERROR('only between 10am and 4pm',16,1)
                END
            END
        ELSE IF DATEPART(DW,GETDATE())=7
            BEGIN
                IF DATEPART(HH,GETDATE()) < 10
                    OR
                    DATEPART(HH,GETDATE()) >= 14
                    BEGIN
                        ROLLBACK
                        RAISERROR('only between 10am and 2pm',16,1)
                    END
                END
            ELSE IF DATEPART(DW,GETDATE())=1
                BEGIN
                    ROLLBACK
                    RAISERROR('sunday not allowed',16,1)
                END
            END

```

Testing :-

wednesday 8am

update emp set sal=1000 where empno = 7369 => ERROR

=> create trigger to not to allow update empno ?

```

CREATE OR ALTER TRIGGER T3
ON EMP
AFTER UPDATE
AS
    IF UPDATE(EMPNO)
        BEGIN
            ROLLBACK
            RAISERROR('empno cannot be updated',16,1)
        END

```

testing :-

update emp set empno=9999 where empno = 7369 => ERROR

24-aug-23

## Magic tables :-

**1 INSERTED**

**2 DELETED**

- => these tables are created and destroyed automatically
- => these table can be accessed only during trigger execution
- => using these tables we can access data affected by dmls.
- => 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 tables.

**INSERT INTO EMP VALUES(100,'A','CLERK',5000,--) => INSERTED**

EMPNO	ENAME	JOB	SAL
100	A	CLERK	5000

**UPDATE EMP SET SAL = 6000 WHERE EMPNO=100 => INSERTED**

EMPNO	SAL
100	6000

**DELETED**

EMPNO	SAL
100	5000

**DELETE FROM EMP WHERE EMPNO = 100 => DELETED**

EMPNO	ENAME	SAL
100	A	6000

**=> 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
```

**=> create trigger to insert details into emp\_resign table when employee resigns ?**

**EMP\_RESIGN**

EMPNO	ENAME	JOB	SAL	HIREDATE	DOR
-------	-------	-----	-----	----------	-----

**CREATE TABLE EMP\_RESIGN**

**(**

```

EMPNO INT,
ENAME VARCHAR(10),
JOB   VARCHAR(10),
SAL   MONEY,
HIREDATE DATE,
DOR   DATE
)

```

```

CREATE OR ALTER TRIGGER T5
ON EMP
AFTER DELETE
AS

```

```

    DECLARE @ENO INT,@NAME VARCHAR(10),@JOB VARCHAR(10)
    DECLARE @SAL MONEY,@HIRE DATE
    SELECT @ENO = EMPNO ,
           @NAME=ENAME,
           @JOB=JOB,
           @SAL=SAL,
           @HIRE=HIREDATE
    FROM DELETED
    INSERT INTO EMP_RESIGN
    VALUES(@ENO,@NAME,@JOB,@SAL,@HIRE,GETDATE())

```

Testing :-

1 delete from emp where empno = 7369

2 select \* from emp\_resign

INSTEAD OF triggers :-

-----

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

AFTER	INSTEAD OF
IF COND	IF COND
BEGIN	RAISERROR
ROLLBACK	ELSE
RAISERROR	DML
END	

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

```

EMP44
ENO      DNO
1        10
2        10
3        10
4        10
5        10 => NOT ALLOWED

```

**CREATE TABLE EMP44(ENO INT,DNO INT)**

**CREATE OR ALTER TRIGGER T6**

**ON EMP44**

**INSTEAD OF INSERT**

**AS**

**DECLARE @ENO INT,@DNO INT,@CNT INT**

**SELECT @ENO = ENO ,@DNO = DNO FROM INSERTED**

**SELECT @CNT = COUNT(\*) FROM EMP44 WHERE DNO = @DNO**

**IF @CNT=4**

**RAISERROR('max 4 emps per dept',16,1)**

**ELSE**

**INSERT INTO EMP44 VALUES(@ENO,@DNO)**

**Testing :-**

**INSERT INTO EMP44 VALUES(1,10)**

**INSERT INTO EMP44 VALUES(2,10)**

**INSERT INTO EMP44 VALUES(3,10)**

**INSERT INTO EMP44 VALUES(4,10)**

**INSERT INTO EMP44 VALUES(5,10) => ERROR**

**=> list of triggers ?**

**SELECT \* FROM SYS.TRIGGERS**

**Dropping triggers :-**

-----

**DROP TRIGGER T1**

**=> if we drop table what about triggers created on table ?**

**ans :- triggers are also dropped**

**SERVER**

**DATABASE**

**TABLE**

**ROWS & COLS**

**CONSTRAINTS**

**INDEXES**

**TRIGGERS**

**VIEW**

**SYNONYM**

**SEQUENCE**

**PROCEDURES**

**FUNCTIONS**

=====

**25-AUG-23**

**Dynamic SQL :-**

-----  
=> SQL commands generated at runtime are called dynamic sql commands

```
ex :-    DROP TABLE EMP (STATIC SQL)

        DECLARE @TNAME VARCHAR(10)
        SET @TNAME = 'EMP'
        DROP TABLE @TNAME    (DYNAMIC SQL)
```

=> Dynamic sql commands are executed by using EXEC procedure

```
EXEC (' Dynamic sql command ')
```

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

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

example 1 :-

=> create procedure to drop table from db ?

```
CREATE OR ALTER PROCEDURE DROP_TABLE
@TNAME VARCHAR(20)
AS
EXEC (' DROP TABLE ' + @TNAME)
```

Execution :-

-----  
EXECUTE DROP\_TABLE 'BILL'

=> create procedure to drop all tables ?

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

EXECUTION :-

```
EXECUTE DROP_ALL_TABLES
```

=====

## **BACKUP & RESTORE :-**

-----

- => DB must be protected from different failures like hardware, software, program**
- => to protect db from different failures sql server supports backup**
- => backup is the process of copying data from db to backup file (.bak)**
- => when actual db is damaged then we can recover db from backup file.**
- => recovering db from backup is called restore.**

### **backup :-**

-----

**1 select the db that you want to take backup**

**BATCH12 => TASK => BACKUP**

**2 select backup type :- FULL / DIFFERENTIAL**

**FULL                    => full db is copied to backup.**

**DIFFERENTIAL   => since last backup whatever changes are made  
only those changes are copied to backup.**

**3 select destination :-     DISK / URL**

**4 enter backup file name for ex d:\naresh\batch12.bak**

**5 ok**

### **command to take backup :-**

-----

**BACKUP DATABASE DB2PM TO DISK = 'D:\NARESH\DB2PM.BAK'**

### **restore :-**

-----

- => select databases => restore database**
- => select source   => device**
- => select backup file name (d:\naresh\db2pm.bak)**
- => ok**

### **command to restore :-**

-----

**=> open master db and execute the following command**

**RESTORE DATABASE DB2PM FROM DISK = 'D:\NARESH\DB2PM.BAK'**

**=> create a procedure to take backup of all databases ?**

**CREATE OR ALTER PROCEDURE backup\_dbs**

```
AS
DECLARE C1 CURSOR FOR  SELECT NAME
                        FROM SYS.DATABASES
                        WHERE DATABASE_ID > 4
DECLARE @DBNAME VARCHAR(20),@FNAME  VARCHAR(100)
OPEN C1
FETCH NEXT FROM C1 INTO @DBNAME
WHILE(@@FETCH_STATUS=0)
BEGIN
    SET @FNAME = 'D:\NARESH\'+'@DBNAME+'.BAK'
    BACKUP DATABASE @DBNAME TO DISK = @FNAME
    FETCH NEXT FROM C1 INTO @DBNAME
END
CLOSE C1
DEALLOCATE C1
```

**execution :-**

-----

```
EXECUTE backup_dbs
```

**Displaying BATCH12.txt.**

