**Database :-**

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

=> db is a organized collection of interrelated data. for a univ db stores

data related to students, courses,faculty etc and a bank db stores data

related to customers,accounts,trans etc.

**Types of Databaes :-**

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

1 OLTP DB (online transaction processing)

2 OLAP DB (online analytical processing)

=> organizations uses OLTP for storing day-to-day transactions

and OLAP for analysis

=>OLTP is for running business and OLAP is used 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 an interface between user and database

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

Evolution of DBMS :-

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

1960 FMS (File Mgmt System)

1970 HDBMS (Hierarchical DBMS)

NDBMS (Network DBMS)

1980 RDBMS (Relational DBMS)

1990 ORDBMS (Object Relational DBMS)

RDBMS :-

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

=> relational data model concepts are introduced by E.F.CODD (Edgar Frank Codd)

=> E.F.CODD introduced 12 rules called codd rules.

=> a db that supports all 12 rules called perfect rdbms software.

Codd’s rules are proposed by a computer scientist named Dr. Edgar F. Codd and he also invent the relational model for database management. These rules are made to ensure data integrity, consistency, and usability. This set of rules basically signifies the characteristics and requirements of a relational database management system ([RDBMS](https://www.geeksforgeeks.org/rdbms-full-form/)). In this article, we will learn about various Codd’s rules.

**Codd’s Rules in DBMS**

**Rule 1: The Information Rule**

All information, whether it is user information or metadata, that is stored in a database must be entered as a value in a cell of a table. It is said that everything within the database is organized in a table layout.

**Rule 2: The Guaranteed Access Rule**

Each data element is guaranteed to be accessible logically with a combination of the table name, primary key (row value), and attribute name (column value).

**Rule 3: Systematic Treatment of NULL Values**

Every Null value in a database must be given a systematic and uniform treatment.

**Rule 4: Active Online Catalog Rule**

The database catalog, which contains metadata about the database, must be stored and accessed using the same relational database management system.

**Rule 5: The Comprehensive Data Sublanguage Rule**

A crucial component of any efficient database system is its ability to offer an easily understandable data manipulation language (DML) that facilitates defining, querying, and modifying information within the database.

**Rule 6: The View Updating Rule**

All views that are theoretically updatable must also be updatable by the system.

**Rule 7: High-level Insert, Update, and Delete**

A successful database system must possess the feature of facilitating high-level insertions, updates, and deletions that can grant users the ability to conduct these operations with ease through a single query.

**Rule 8: Physical Data Independence**

Application programs and activities should remain unaffected when changes are made to the physical storage structures or methods.

**Rule 9: Logical Data Independence**

Application programs and activities should remain unaffected when changes are made to the logical structure of the data, such as adding or modifying tables.

**Rule 10: Integrity Independence**

Integrity constraints should be specified separately from application programs and stored in the catalog. They should be automatically enforced by the database system.

**Rule 11: Distribution Independence**

The distribution of data across multiple locations should be invisible to users, and the database system should handle the distribution transparently.

**Rule 12: Non-Subversion Rule**

If the interface of the system is providing access to low-level records, then the interface must not be able to damage the system and bypass security and integrity constraints.

Information Rule :-

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

=> according to information rule data must be organized in tables i.e. rows and columns

CUST

CID CNAME CITY => column / field / attribute

10 SACHIN MUM

11 RAHUL DEL

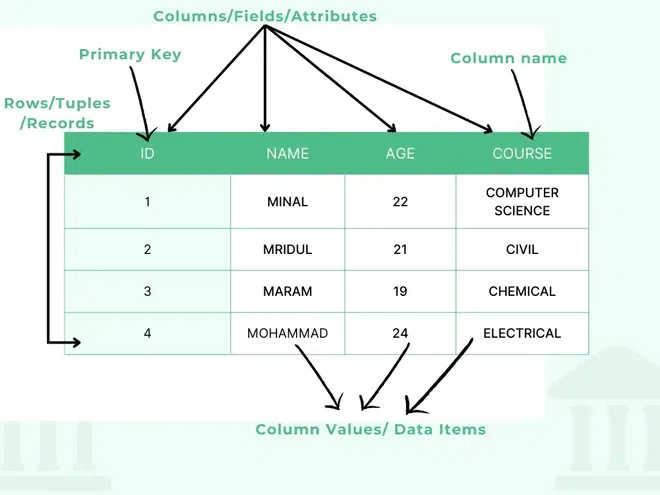
12 VIJAY HYD => row/record/tuple

Database = Collection of tables

Table = Collection of rows & cols

Row = Collection of field values

Column = Collection of values of one field



=> every table must contain primary key to uniquely identify

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

RDBMS features :-

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

1 easy to access and manipulate data

2 less redundency (duplication of data)

3 more security

4 supports data integrity i.e. data quality

5 supports data sharing

6 supports transactions(ACID properties)

A atomocity

C consistency

I isolation

D durability

RDBMS softwares :- (SQL databaes)

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

SQL SERVER from microsoft

ORACLE from oracle corp

MYSQL from oracle corp

DB2 from ibm

RDS from amazon

POSTGRESQL from postgresql

NoSQL Databases :-

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

MongoDB

cassandra

ORDBMS :-

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

=> ordbms is combination of RDBMS & OOPS

ORDBMS = RDBMS + OOPS (reusability)

=> rdbms doesn't supports reusability but ordbms supports reusability.

Examples :-

SQL SERVER

ORACLE

POSTGRESQL

what is db ?

what is dbms ?

what is rdbms ?

what is ordbms ?

10-jun-24

DB Development Life Cycle :-

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

Analyse

Design

Develop

Test

Deploy / Implement

Maintain

Design :-

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

=> Designing db means designing tables

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

1 ER Model (Entity Relationship)

2 Normalization

Development :-

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

=> DB is developed by Developers & DBAs (DB Admin)

=> DB is developed by using any rdbms tool like SQL SERVER.

Roles and Responsibilities of Developer and DBA

|  |  |  |
| --- | --- | --- |
| **Developer** | | **DBA** |
| creating tables | | installation of sql server |
| creating views | | creating databases |
| creating synonyms | | creating logins |
| creating sequences | | db backup & restore |
| creating indexes | | db export & import |
| creating procedures | | db upgradation & migration |
| creating functions | | performance tuning |
| creating triggers | writing queries | |

sql server 2012 =====> sql server 2019 upgradation

mysql ===========> sql server migration

Testing :-

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

=> DB is tested by QA (Quality Assurance) team by using

1 manual

2 automation tools like selenium

Deploy / Implementation :-

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

=> copying db from dev server to prod server is called deploy or implementation

and after deploying db end users can use db for day-to-day transactions.

=================================================================

SQL SERVER

===========

=> sql server is basically a rdbms product from microsoft and also supports

ordbms features and used to create and to manage database.

=> sql server can be used for both DB Development & DB Administration.

sql server versions :-

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

version year

SQL SERVER 1.1 1991

SQL SERVER 4.2 1993

SQL SERVER 6.0 1995

SQL SERVER 6.5 1996

SQL SERVER 7.0 1998

SQL SERVER 2000 2000

SQL SERVER 2005 2005

SQL SERVER 2008 2008

SQL SERVER 2012 2012

SQL SERVER 2014 2014

SQL SERVER 2016 2016

SQL SERVER 2017 2017

SQL SERVER 2019 2019

SQL SERVER 2022 2022

11-jun-24

CLIENT / SERVER Architecture :-

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

1 server

2 client

server :-

-----------

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

=> inside the server sql server manages

1 DB

2 INSTANCE

=> DB is created in hard disk and acts as permanent storage

=> INSTANCE is created in ram and acts as temporary storage

client :-

-----------

=> client is also a system from where users can

1 connects to server

2 submit requests

3 receives response

client tool :-

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

SSMS (Sql Server Management Studio)

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

SQL :-

----------

=> structured query language

=> 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 language was

**"sequel"** and later it is renamed to SQL.

=> common to all relational database softwares.

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

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

USER---MYSQLWORK-----------SQL----------------------MYSQL-------------DB

BENCH

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

DDL (DATA DEFINITION LANG)

DML (DATA MANIPULATION LANG)

DQL (DATA QUERY LANG)

TCL (TRANSACTION CONTROL LANG)

DCL (DATA CONTROL LANG)

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

DATA & DATA DEFINITION :-

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

EMPID ENAME SAL => DATA DEFINITION / METADATA

1 A 5000 => DATA

download & install :-

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

1 SQL server

2 ssms

SQL server :-

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

download :-

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

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

installation :-

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

https://www.mssqltips.com/sqlservertip/7313/install-sql-server-2022/

ssms :-

---------

download :-

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

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

12-jun-24

How to connect to SQL server :-

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

=> open ssms and enter following details

SERVER TYPE :- DB ENGINE

SERVER NAME :- DESKTOP-G2DM7GI

AUTHENTICATION :- SQL SERVER Authentication

LOGIN :- SA (system admin)

PASSWORD :- 123

=> click connect

=> by default, SQL server comes with following 4 databases called system databases

1 master

2 model

3 msdb

4 tempdb

creating new database :-

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

=> In object explorer select Databases => New Database

Enter Database Name :- DB11AM

=> click OK

=> a DB is created with following two files

1 DATA

2 LOG

=> DATA file stores data and log file stores operations

NAME TYPE SIZE AUTOGROWTH PATH

DB11AM DATA 8 MB 64 MB C:\----

DB11AM\_LOG LOG 8 MB 64 MB C:\---

path :-

-----------

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

DB11AM.MDF

DB11AM\_LOG.LDF

**mdf => master data file**

**ldf => log data file**

command to create new database :-

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

=> open master db as follows

Databases

system databases

master => New Query

=> type the following query to create new database

CREATE DATABASE salesDB

Datatypes in SQL SERVER :-

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

=> a datatype specifies

1 type of the data allowed in a column

2 amount of memory allocated for column

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **DATATYPES** | | | | | | |
| **CHAR** | | **INTEGER** | **FLOAT** | **CURRENCY** | **DATE** | **BINARY** |
| **ASCII** | **UNICODE**(n-National) |
| char | nchar | tinyint | numeric(p,s) | smallmoney | date | Binary |
| varchar | nvarchar | smallint | decimal(p,s) | money | time | varbinary |
| varchar(max) | nvarchar(max) | int |  |  | datetime | varbinary(max) |
|  |  | bigint |  |  |  |  |
|  |  | numeric(p) |  |  |  |  |

**char(size) :-**

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

=> allows character data up to 8000 chars

=> recommended for fixed length char columns

ex :- NAME CHAR(10)

SACHIN - - - -

wasted

RAVI - - - - - -

wasted

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

ex :- GENDER CHAR(1)

M

F

STATE\_CODE CHAR(2)

AP

TG

**varchar(size) :-**

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

=> allows character data up to 8000 chars

=> recommended for variable length fields

ex :- NAME VARCHAR(10)

SACHIN - - - -

released

=> In varchar extra bytes are released

**NOTE :-** char,varchar allows ascii chars (256 chars) that includes a-z,A-Z,0-9

and special chars.

ex :- PANNO CHAR(10)

VEHNO CHAR(10)

EMAILID VARCHAR(30)

**VARCHAR(max) :-**

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

=> allows character data up to **2GB**

ex :- TEXT VARCHAR(max)

1024 bytes = 1 KB

1024 KB = 1 MB

1024 MB = 1 GB

1024 GB = 1 TB

1024 TB = 1 PB



**nchar/nvarchar/nvarchar(max) :- ( n => national)**

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

=> allows Unicode chars (65536 chars) that includes all ASCII chars and chars belongs to different languages

13-JUN-24

**Integer Types :-**

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

=> allows numbers without decimal

|  |  |  |  |
| --- | --- | --- | --- |
| Tinyint | | 1 | 0 to 255 |
| Smallint | | 2 | -32768 to 32767 (-2^15 to 2^15-1) |
| Int | | 4 | -2^31 to 2^31-1 (-2147483648 to 2147483647) |
| bigint | 8 | | -2^63 to 2^63-1 (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807) |

ex :- age tinyint

empid smallint

custid int

**Numeric(p) :-**

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

=> allows numbers without decimal up to 38 digits

ex :- empid NUMERIC(4)

10

100

1000

10000 => not allowed

phone NUMERIC(10)

aadharno NUMERIC(12)

accno NUMERIC(15)

**numeric(p,s) / decimal(p,s) :-**

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

=> allows numbers with decimal

p => precision => total no of digits allowed

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

ex :- salary DECIMAL(7,2)

5000

5000.55

50000.55

500000.55 => not allowed

5000.5689 => allowed => 5000.57

5000.5637 => allowed => 5000.56

5000.5657 => allowed => 5000.57

savg DECIMAL(5,2)

**CURRENCY types :-**

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

=> currency types are used for fields related to money

smallmoney 4 -214748.3648 **to** 214748.3647

money 8 -922,337,203,685,477.5808 **to** 922,337,203,685,477.5807

ex :- salary smallmoney

bal money

**DATE & TIME :-**

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

DATE => allows only date

TIME => allows only time

DATETIME => allows both date & time

=> default date format in SQL server is yyyy-mm-dd

=> default time format is hh:mi:ss

ex :- DOB DATE

2003-10-5

LOGIN TIME

10:00:00

TXNDT DATETIME

2024-06-13 9:30:30

**Binary Types :-**

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

=> allows multimedia objects like audio, video, images

binary => allows binary data up to 8000 bytes

varbinary => allows binary data up to 8000 byes

varbinary(max) => allows binary data up to 2GB

ex :- photo binary(5000)

photo varbinary(5000)

video varbinary(max)

===============================================================

CREATING TABLES IN SQL SERVER DB:-

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

CREATE TABLE <tabname>

(

colname datatype(size) ,

colname datatype(size),

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

)

**Rules :-**

----------

1 table name should start with alphabet

2 name should not contain spaces & special chars but allows \_(underscore), $(doller),#(pound)

3 name can be upto 128 chars

**4 table can have 1024 cols**

**5 no of rows unlimited**

ex :- 123emp invalid

emp 123 invalid

emp\*123 invalid

emp\_123 valid

=> create table with following structure ?

EMP

EMPID ENAME JOB SAL HIREDATE DEPT

CREATE TABLE EMP

(

EMPID SMALLINT,

ENAME VARCHAR(10),

JOB VARCHAR(10),

SAL SMALLMONEY,

HIREDATE DATE,

DEPT VARCHAR(10)

)

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

SP\_HELP :- sp => stored procedure

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

=> command to see the structure of the table

SP\_HELP <tabname>

ex :-

SP\_HELP EMP

EMPID smallint 2

ENAME varchar 10

JOB varchar 10

SAL smallmoney 4

HIREDATE date 3

DEPT varchar 10

**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 :-**

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

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

ex :-

INSERT INTO EMP VALUES(100,'sachin','clerk',4000,'2024-06-13','hr')

INSERT INTO EMP VALUES(101,'virat','manager',8000,getdate(),'it')

14-jun-24

**inserting multiple rows :-**

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

INSERT INTO EMP VALUES(102,'david','analyst',6000,'2020-08-15','sales') ,

(103,'anand','clerk',5000,'2018-04-20','it'),

(104,'rahul','manager',10000,'2019-09-12','sales')

create table emp(empid int, ename varchar(50),job varchar(10), sal smallmoney, hiredate date, dept varchar(10));

sp\_help emp;

select \* from emp;

insert into emp values(101,'Sachin','Clerk',4000,'2024-06-13','HR');

insert into emp values(102,'Virat','Manager',8000,getdate(),'IT');

insert into emp values(103,'David','Analyst',6000,'2020-08-15','SALES'),

(104,'Anand','Clerk',5000,'2018-04-20','IT'),

(105,'Rahul','Manager',10000,'2019-09-12','SALES');

insert into emp values(106,'Vijay',NULL, NULL,getdate(),'IT');

insert into emp (empid,ename,hiredate,dept)

values(107,'Pravin',getdate(),'IT');

**inserting nulls :-**

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

=> a null means blank or empty

=> it is not equal to 0 or space

=> nulls can be inserted in two ways

**method 1 :-**

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

INSERT INTO EMP VALUES(105,'vijay',null,null,'2021-02-15','hr')

**method 2 :-**

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

INSERT INTO EMP(EMPID,ENAME,HIREDATE,DEPT)

VALUES(106,'pravin',getdate(),'it')

remaining two fields job,sal are filled with nulls

==============================================================

**Operators in SQL SERVER :-**

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

Arithmetic Operators => + - \* / %

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

Logical Operators => AND OR NOT

Special Operators => BETWEEN

IN

LIKE

IS

ANY

ALL

EXISTS

PIVOT

Set Operators => UNION

UNION ALL

INTERSECT

EXCEPT

**Displaying Data :-**

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

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

=> we can display all rows or specific rows

=> we can display all cols or specific cols

syntax :- SELECT columns / \* FROM tablename

SQL = ENGLISH

QUERIES = SENTENCES

CLAUSES = WORDS

\* => all columns

ex :-

=> display employee names and salaries?

SELECT ENAME,SAL FROM EMP

|  |  |
| --- | --- |
| ename | sal |
| Sachin | 4000 |
| Virat | 8000 |
| David | 6000 |
| Anand | 5000 |
| Rahul | 10000 |
| Vijay | NULL |
| Pravin | NULL |

=> display employee names,job,sal and hiredates ?

SELECT ENAME,JOB,SAL,HIREDATE FROM EMP

|  |  |  |  |
| --- | --- | --- | --- |
| **ENAME** | **JOB** | **SAL** | **HIREDATE** |
| Sachin | Clerk | 4000 | 13-06-2024 |
| Virat | Manager | 8000 | 27-07-2024 |
| David | Analyst | 6000 | 15-08-2020 |
| Anand | Clerk | 5000 | 20-04-2018 |
| Rahul | Manager | 10000 | 12-09-2019 |
| Vijay | NULL | NULL | 27-07-2024 |
| Pravin | NULL | NULL | 27-07-2024 |

=> display all the data from emp table ?

SELECT \* FROM EMP

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT |
| 107 | Pravin | NULL | NULL | 27-07-2024 | IT |

**WHERE clause :-**

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

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

=> where clause is used to filter out the rows based on the condition(s).

SELECT columns / \*

FROM tablename

WHERE condition

**condition :-**

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

COLNAME OP VALUE

=> a **condition** is a relational expression that **returns true / false**

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

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

**ex :-**

=> display employee details whose id = 103 ?

SELECT \* FROM EMP WHERE EMPID = 103

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPID | ENAME | JOB | SAL | HIREDATE | DEPT |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |

=> display employee details whose name = vijay ?

SELECT \* FROM EMP WHERE ENAME = 'vijay'

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EMPID | ENAME | JOB | SAL | HIREDATE | DEPT |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT |

**Note :** Case doesn’t matter 'vijay' and 'Vijay' both are same

=> employee details earning more than 5000 ?

SELECT \* FROM EMP WHERE SAL > 5000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

=> employees joined after 2020 ?

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

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT |
| 107 | Pravin | NULL | NULL | 27-07-2024 | IT |

=> employees joined before 2020 ?

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

=> employees not working for sales dept ?

SELECT \* FROM EMP WHERE DEPT <> 'sales'

15-jun-24

**Compound Condition :-**

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

=> multiple conditions combined with AND / OR operators are 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 |

=> employees working as clerk, manager ?

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

=> employees whose id = 101,103,105 ?

SELECT \*

FROM EMP

WHERE EMPID=101 OR EMPID = 103 OR EMPID=105

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

=> display sales dept manager details?

SELECT \*

FROM EMP

WHERE DEPT = 'sales' AND JOB = 'manager'

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Empid** | **Ename** | **Job** | **Sal** | **Hiredate** | **Dept** |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

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

SELECT \*

FROM EMP

WHERE SAL > 5000 AND SAL < 10000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |

=> employees joined in 2020 ?

SELECT \*

FROM EMP

WHERE HIREDATE >= '2020-01-01'

AND

HIREDATE <= '2020-12-31'

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |

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

SELECT \*

FROM EMP

WHERE JOB='clerk'

OR

JOB='manager'

AND

SAL>5000

=>**Note:** above query returns clerks earning less than 5000 because operator AND

has got more priority than operator OR , so SAL>5000 applied only to

manager and it is not applied to clerk, so to overcome this problem use ( ).

SELECT \*

FROM EMP

WHERE (

JOB='clerk'

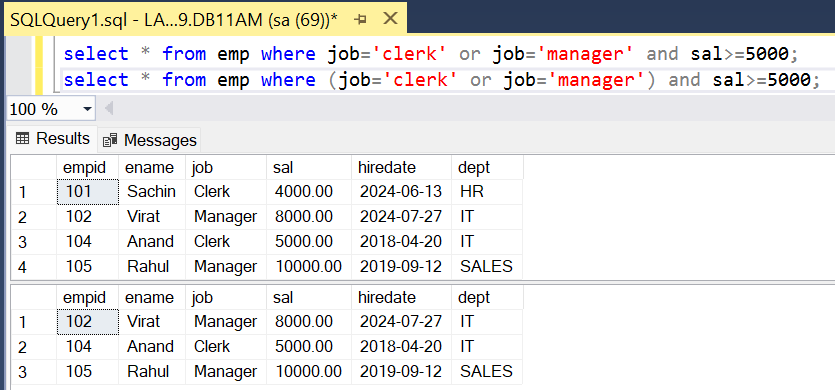
OR

JOB='manager'

)

AND

SAL>5000



=>

STUDENT

SNO SNAME S1 S2 S3

1 A 80 90 70

2 B 30 60 50

CREATE TABLE STUDENT

(

SID INT,

SNAME VARCHAR(10),

S1 TINYINT,

S2 TINYINT,

S3 TINYINT

)

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

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

list of students who are passed?

SELECT \* FROM STUDENT WHERE S1>=35 AND S2>=35 AND S3>=35

list of students who are failed?

SELECT \* FROM STUDENT WHERE S1<35 OR S2<35 OR S3<35

**IN operator:-**

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

=> use IN operator for list comparison.

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

=> IN operator is used whenever we have to filter the row based on different values of same column.

=> It reduces the no of comparisons and joining of conditions using operators.

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

WHERE COLNAME IN (V1,V2,V3,---) => Valid

=> employees whose id =101,103,105 ?

SELECT \* FROM EMP WHERE EMPID **IN** (101,103,105)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

=> employees working as clerk,manager ?

SELECT \* FROM EMP WHERE JOB **IN** ('clerk','manager')

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

=> employees not working for dept it,hr ?

SELECT \* FROM EMP WHERE DEPT **NOT IN** ('hr','it')

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

**BETWEEN operator :-**

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

=> use between operator for range comparison i.e. between two values

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

=> employees earning between 5000 and 10000 ?

SELECT \*

FROM EMP

WHERE SAL **BETWEEN** 5000 **AND** 10000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |

=> employees joined in 2020 ?

SELECT \*

FROM EMP

WHERE HIREDATE **BETWEEN** '2020-01-01' **AND** '2020-12-31'

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES |

=> not joined in 2020 ?

WHERE HIREDATE **NOT BETWEEN** '2020-01-01' **AND** '2020-12-31'

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT |
| 107 | Pravin | NULL | NULL | 27-07-2024 | IT |

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

and not joined in 2020 and not working for dept hr,sales ?

SELECT \*

FROM EMP

WHERE JOB IN ('clerk','manager')

AND

SAL BETWEEN 5000 AND 10000

AND

HIREDATE NOT BETWEEN '2020-01-01' AND '2020-12-31'

AND

DEPT NOT IN ('hr','sales')

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT |

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

PRODUCTS

prodid pname price category brand

SELECT \*

FROM PRODUCTS

WHERE BRAND IN ('samsung','redmi','realme')

AND

PRICE BETWEEN 10000 AND 20000

AND

CATEGORY='mobiles'

17-JUN-24

**IMP Note:**

SELECT \*

FROM EMP

WHERE SAL BETWEEN 10000 AND 5000

A ERROR

B RETURNS NO ROWS

C RETURNS ROWS

D NONE

ANS :- B

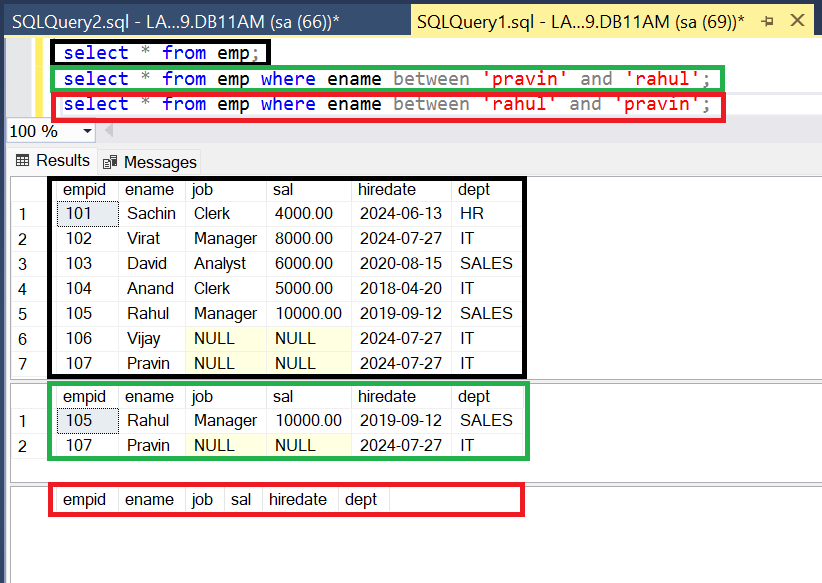
Internally :

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

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

**NOTE :-**

=> **use between with lower and upper but not with upper and lower**



**LIKE operator:-**

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

=> use LIKE operator for pattern comparison

ex :- name starts with 's'

name ends with 'd'

name contains 'a'

Syntax:

WHERE COLNAME LIKE 'PATTERN'

=> pattern contains alphabets, digits, wildcard chars

=> Brackets in Pattern not mandatory, ' ' single quote is mandatory.

**Wildcard Chars :-**

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

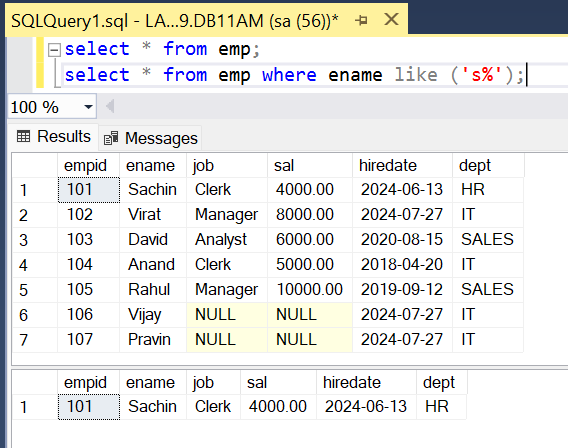
Using wildcard characters makes the LIKE operator more flexible than using the = and != string comparison operators. If any one of the arguments isn't of character string data type, the SQL Server Database Engine converts it to character string data type, if it's possible.

% => 0 or many chars

\_ => exactly 1 char

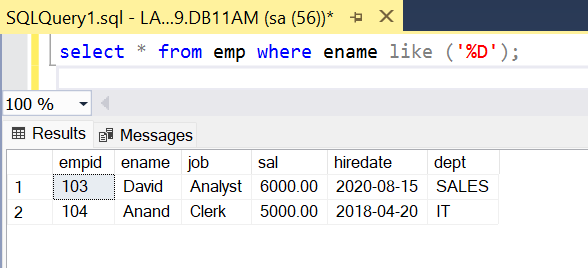
=> employees name starts with 's' ?

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



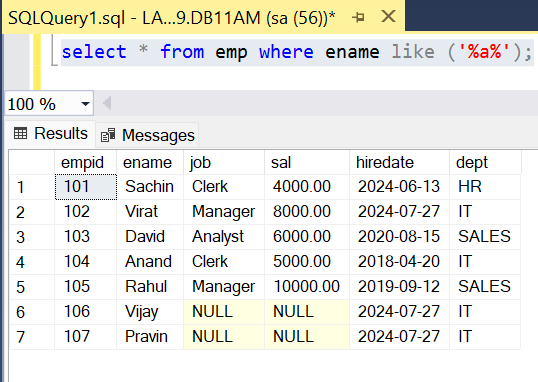
=> employees name ends with 'd' ?

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



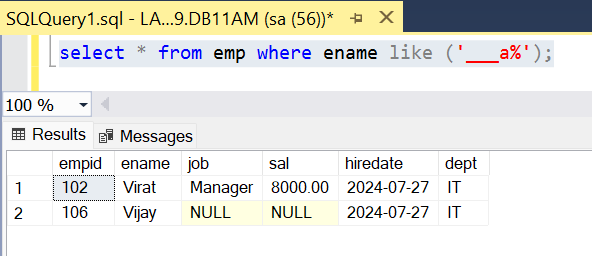
=> employees name contains 'a' ?

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



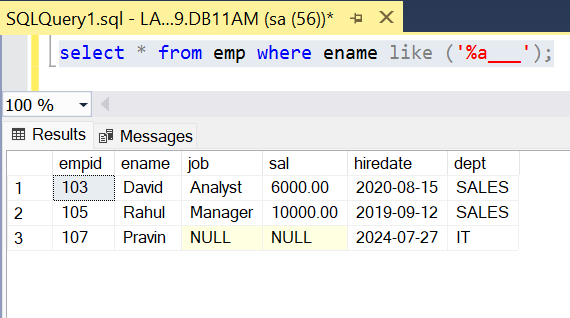
=> employees where 'a' is the 3rd char in their name ?

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



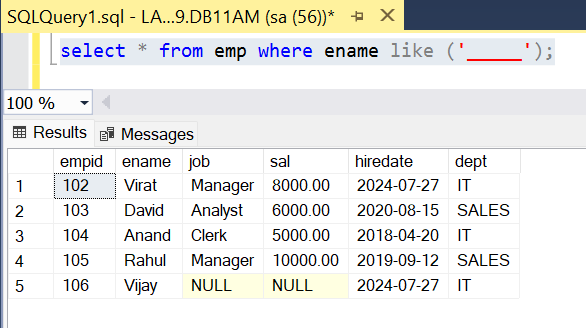
=> employees where 'a' is the 4th char from last ?

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



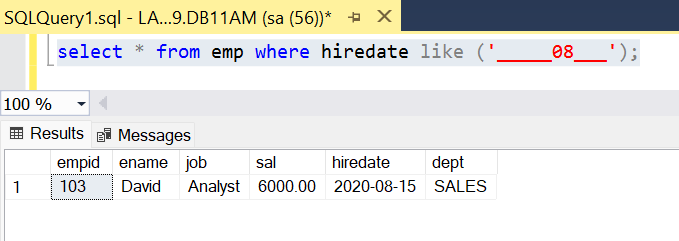
=> name contains 5 chars ?

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



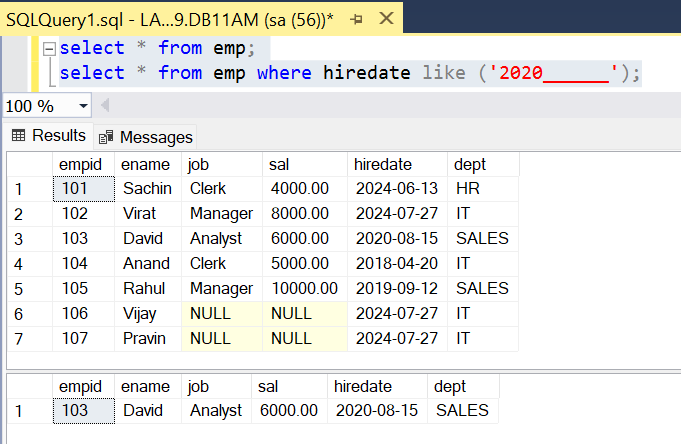
=> employees joined in aug month ? YYYY-MM-DD

SELECT \* FROM EMP WHERE **HIREDATE** LIKE '\_\_\_\_\_08\_\_\_'



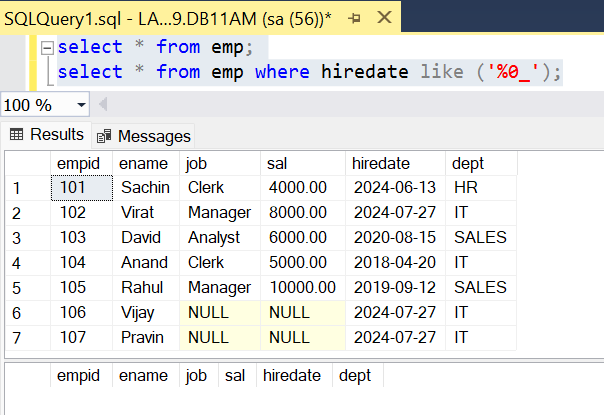
=> employees joined in 2020 year ?

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



=> employees joined in 1st 9 days of any of month any year ?

SELECT \* FROM EMP WHERE HIREDATE LIKE '%0\_'

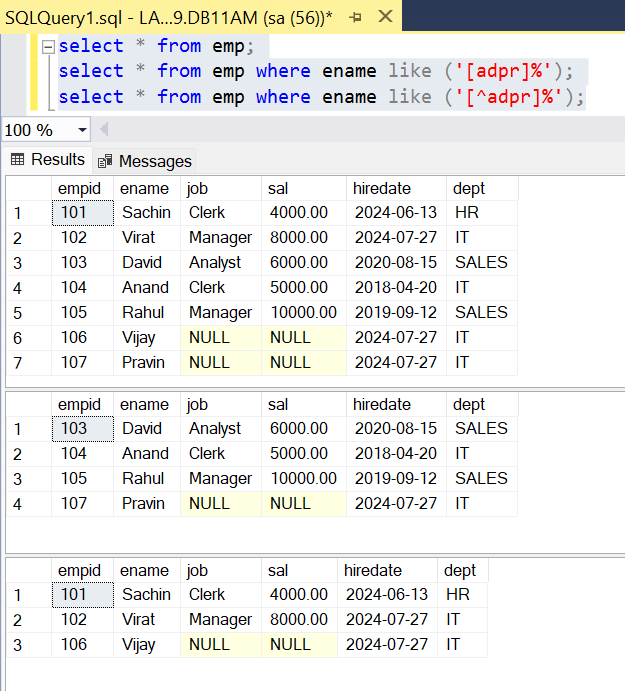


=> employees name starts with 'a','d','p','r' ?

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

=> employees name do not starts with 'a','d','p','r' ?

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



=> employees name starts between 'a' and 'p' ?

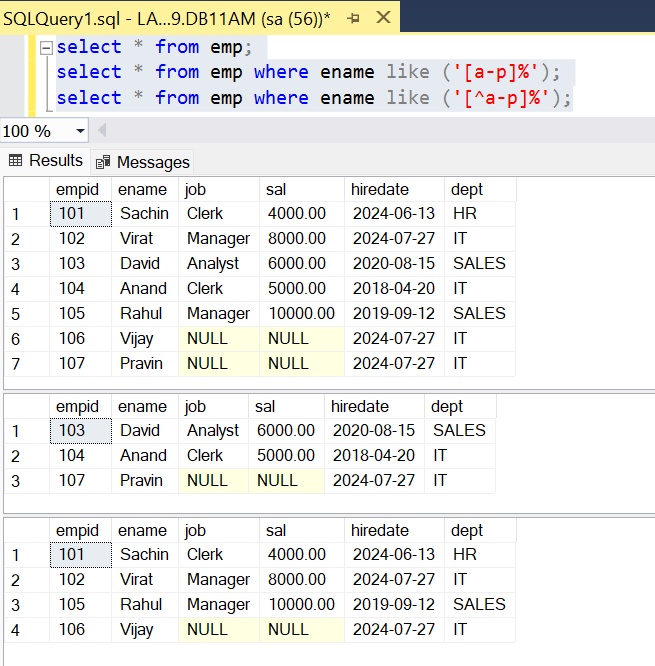
SELECT \* FROM EMP WHERE ENAME LIKE '[a-p]%'

=> employees name do not starts between 'a' and 'p' ?

SELECT \* FROM EMP WHERE ENAME LIKE '[^a-p]%'

or

SELECT \* FROM EMP WHERE ename **not** like ('[a-p]%');



=>

CUST

CID CNAME

10 sachin\_tendulkar

11 virat%kohli

12 mahendra\_singh\_dhoni

Create above table:

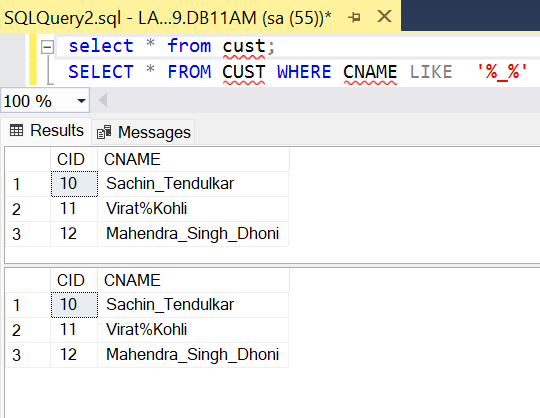
create table cust (CID INT, CNAME VARCHAR(20))

Insert Records:

insert into cust values(10,'Sachin\_Tendulkar'),(11,'Virat%Kohli'),(12,'Mahendra\_Singh\_Dhoni')

=> employees name contains "\_" ?

SELECT \* FROM CUST WHERE CNAME LIKE '%\_%'

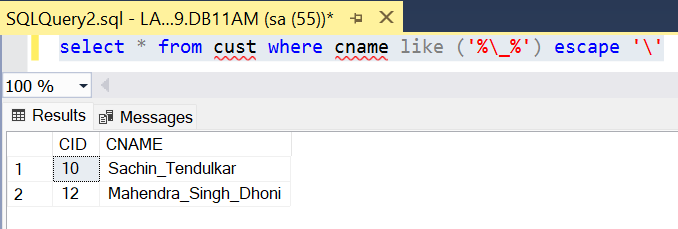


above query displays all cust records because '\_' is not treated as search char and

it is treated as wildcard char , to overcome this problem use escape char.

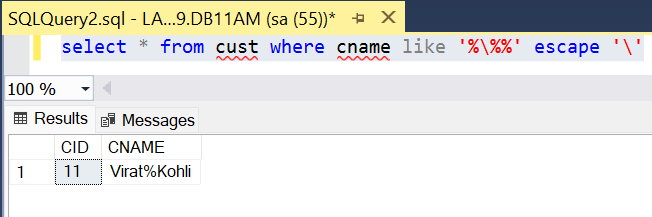
SELECT \* FROM CUST WHERE CNAME LIKE '%\\_%' ESCAPE '\'

the char that immediately follows "\" is not treated as wildcard char and it is treated as normal char



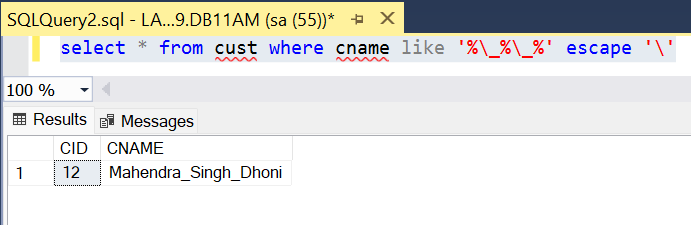
=> name contains "%" ?

SELECT \* FROM CUST WHERE CNAME LIKE '%\%%' ESCAPE '\'



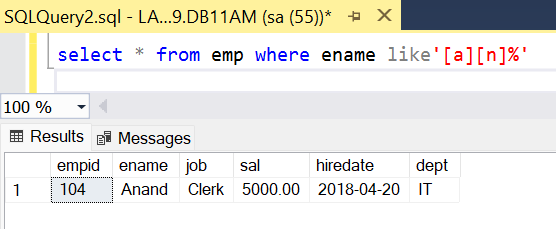
=> name contains 2 "\_" ?

SELECT \* FROM CUST WHERE CNAME LIKE '%\\_%\\_%' ESCAPE '\'



=> Filter results for description with first character A and second character N

select \* from emp where ename like'[a][n]%'



\*alter table emp add email varchar(20)

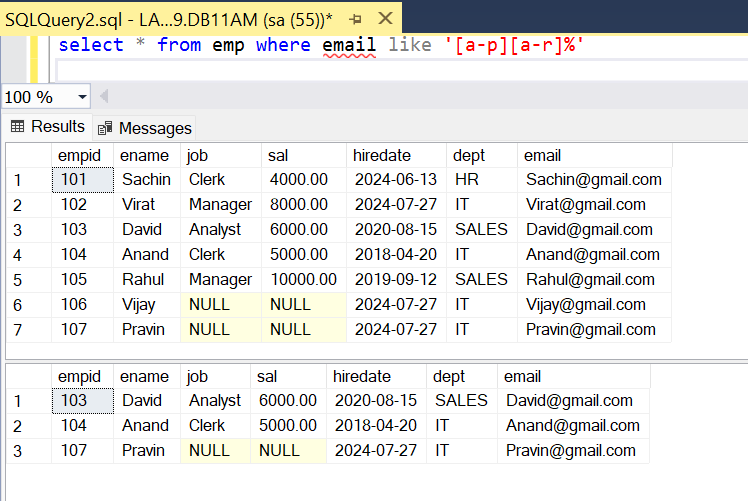
\*update emp set email=ename+'@gmail.com'

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Empid** | **Ename** | **Job** | **Sal** | **Hiredate** | **Dept** | **Email** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR | Sachin@gmail.com |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT | Virat@gmail.com |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES | David@gmail.com |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT | Anand@gmail.com |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES | Rahul@gmail.com |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT | Vijay@gmail.com |
| 107 | Pravin | NULL | NULL | 27-07-2024 | IT | Pravin@gmail.com |

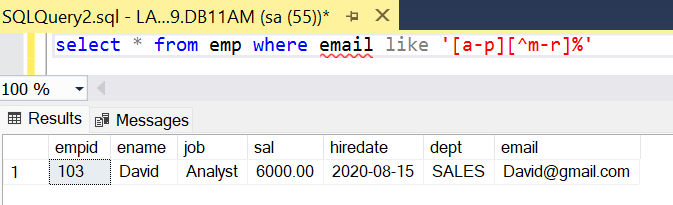
Select The emp with first character should be from A and D alphabets the second character should be from F and L alphabet.

select \* from emp

select \* from emp where email like '[a-p][a-r]%'



select \* from emp where email like '[a-p][^m-r]%'



18-jun-24

**IS operator:-**

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

=> In databases, NULL is unknown, not applicable or missing information, therefore, you cannot use the comparison operators (=, >,<, etc.,) to check whether a value is NULL or not. For example,

The expression A = NULL, B <> NULL or NULL = NULL returns NULL because NULL values cannot be compared.

Fortunately, SQL provides the IS operator to check whether a value is NULL.

=> use IS operator for NULL comparison.

WHERE COLNAME IS NULL

WHERE COLNAME IS NOT NULL

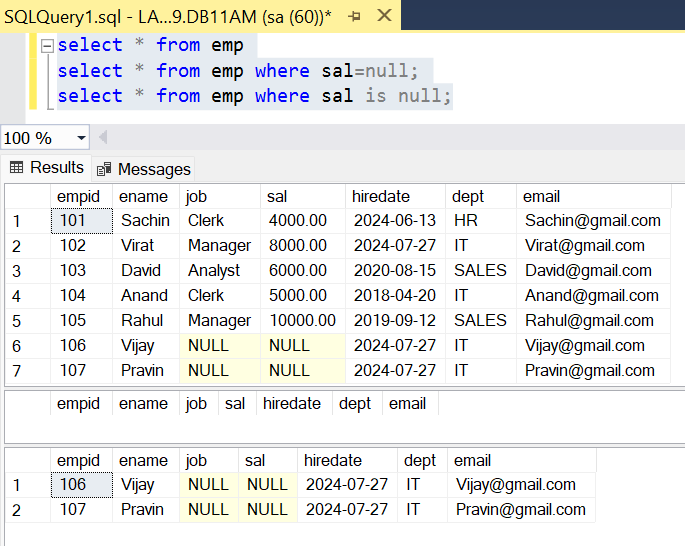
=> employees who are not earning salary?

SELECT \* FROM EMP WHERE SAL IS NULL

select \* from emp To show all Records

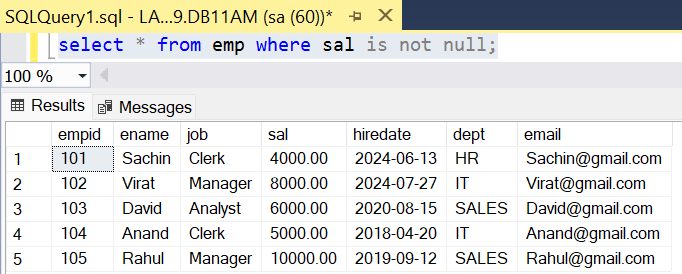
select \* from emp where sal=null; Invalid

select \* from emp where sal is null; Valid



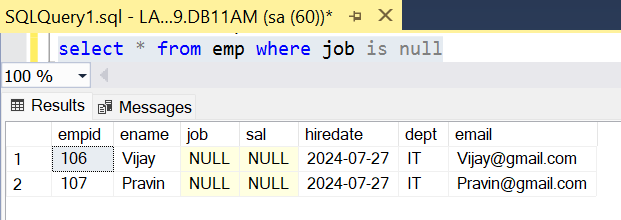
=> employees earning salary ?

SELECT \* FROM EMP WHERE SAL IS NOT NULL



=> employees not assigned with any job ?

SELECT \* FROM EMP WHERE JOB IS NULL



**Summary:-**

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

WHERE COLNAME BETWEEN V1 AND V2

WHERE COLNAME LIKE 'PATTERN'

WHERE COLNAME IS NULL

**ALIAS :-**

-----------

=> SQL aliases are **used to give a temporary Name to table, or a column in a table.**

=> Aliases are often used to make **column names more readable**.

=> **An alias only exists for the duration of that query**.

=> An alias is created with the **AS** keyword.

=> Alias means another name or alternative name

=> used to change column heading

COLNAME/EXPR [AS] ALIAS

**Ex :-**

=> display ENAME ANNUAL SAL ?

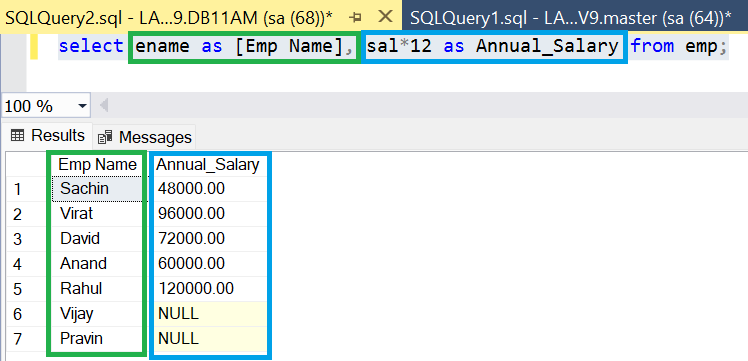
SELECT ENAME,SAL\*12 AS ANNSAL

FROM EMP

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

FROM EMP

=> select ename as [Emp Name], sal\*12 as Annual\_Salary from emp;



=> display ENAME SAL HRA DA TAX TOTSAL ?

HRA = house rent allowance = 20% on sal

DA = dearness allowance = 30% on sal

TAX = 10% on sal

TOTSAL = SAL + HRA + DA - TAX

select ename as [EMP Name],

sal as ***[Basic Salary]***,

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 [***Total Salary***] from emp

**Output:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP Name** | **Basic Salary** | **HRA** | **DA** | **TAX** | **Total Salary** |
| Sachin | 4000 | 800 | 1200 | 03-02-1901 | 6400 |
| Virat | 8000 | 1600 | 2400 | 10-03-1902 | 12800 |
| David | 6000 | 1200 | 1800 | 22-08-1901 | 9600 |
| Anand | 5000 | 1000 | 1500 | 14-05-1901 | 8000 |
| Rahul | 10000 | 2000 | 3000 | 26-09-1902 | 16000 |
| Vijay | NULL | NULL | NULL | NULL | NULL |
| Pravin | NULL | NULL | NULL | NULL | NULL |

=> employees earning more than 1 lac annual salary ?

SELECT \*

FROM EMP

WHERE SAL\*12 > 100000

==============================================================

**ORDER BY clause :-**

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

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

SELECT columns

FROM tabname

[WHERE cond]

ORDER BY colname ASC / DESC ,------

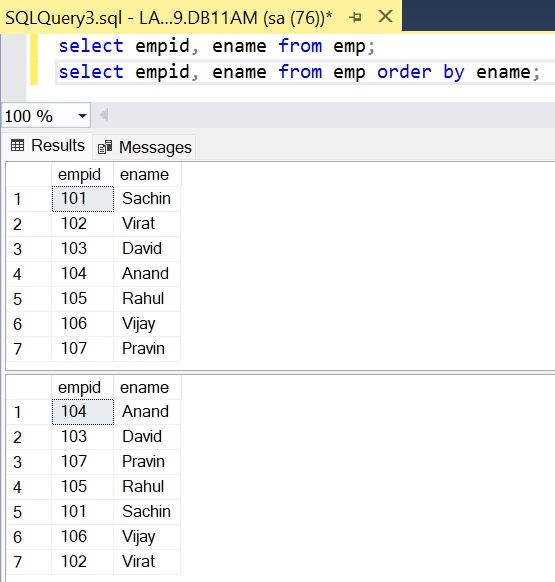
Ex :-

=> arrange employee list name wise ascending?

SELECT \*

FROM EMP

ORDER BY ENAME ASC



=> arrange employee list sal wise desc ?

SELECT \*

FROM EMP

ORDER BY SAL DESC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EmpID** | **Name** | **Sal** |  | **EmpID** | **Name** | **Sal** |
| 1 | A | 3000 |  | 2 | B | 6000 |
| 2 | B | 6000 | ====> | 4 | D | 5000 |
| 3 | C | 4000 |  | 3 | C | 4000 |
| 4 | D | 5000 |  | 1 | A | 3000 |

=> arrange employee hiredate wise ? employee who joined first display first ?

SELECT \*

FROM EMP

ORDER BY HIREDATE ASC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Empid** | **Ename** | **Job** | **Sal** | **Hiredate** | **Dept** | **Email** |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT | Anand@gmail.com |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES | Rahul@gmail.com |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES | David@gmail.com |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR | Sachin@gmail.com |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT | Virat@gmail.com |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT | Vijay@gmail.com |
| 107 | Pravin | NULL | NULL | 27-07-2024 | IT | Pravin@gmail.com |

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

create table emp1 (empid int, ename varchar(10),sal money,deptid int)

insert into emp1 values(1,'A',3000,20),

(2,'B',5000,10),

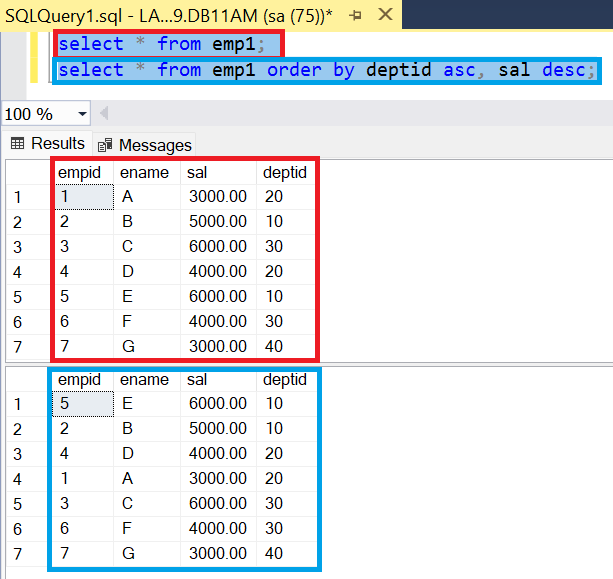
(3,'C',6000,30),

(4,'D',4000,20),

(5,'E',6000,10),

(6,'F',4000,30),

(7,'G',3000,40)



=> find out employees highest and least experienced in each dept ?

SELECT EMPNO,ENAME,HIREDATE,DEPTNO

FROM EMP

ORDER BY DEPTNO ASC ,HIREDATE ASC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** | **EMAIL** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR | Sachin@gmail.com |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT | Virat@gmail.com |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT | Vijay@gmail.com |
| 107 | Pravin | NULL | NULL | 27-07-2024 | IT | Pravin@gmail.com |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT | Anand@gmail.com |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES | David@gmail.com |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES | Rahul@gmail.com |

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

create table student1(rollno int, sname varchar(10),Maths tinyint, phys tinyint, chem tinyint);

insert into student1 values(1,'A',80,90,70),

(2,'B',60,70,50),

(3,'C',90,70,80),

(4,'D',90,80,70);

STUDENT1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLLNO** | **SNAME** | **MATHS** | **PHYS** | **CHEM** |
| 1 | A | 80 | 90 | 70 |
| 2 | B | 60 | 70 | 50 |
| 3 | C | 90 | 70 | 80 |
| 4 | D | 90 | 80 | 70 |

SELECT \* , (MATHS+PHYS+CHEM)/3 AS AVG FROM STUDENT1 ORDER BY (MATHS+PHYS+CHEM)/3 DESC, MATHS DESC, PHYS DESC;

**OUTPUT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ROLLNO** | **SNAME** | **MATHS** | **PHYS** | **CHEM** | **AVG** |
| 4 | D | 90 | 80 | 70 | 80 |
| 3 | C | 90 | 70 | 80 | 80 |
| 1 | A | 80 | 90 | 70 | 80 |
| 2 | B | 60 | 70 | 50 | 60 |

**19-jun-24**

=> employees working as clerk, manager and arrange list sal wise desc ?

SELECT \*

FROM EMP

WHERE JOB IN ('CLERK','MANAGER')

ORDER BY SAL DESC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** | **EMAIL** |
| 105 | Rahul | Manager | 18-05-1927 | 12-09-2019 | SALES | Rahul@gmail.com |
| 102 | Virat | Manager | 25-11-1921 | 27-07-2024 | IT | Virat@gmail.com |
| 104 | Anand | Clerk | 08-09-1913 | 20-04-2018 | IT | Anand@gmail.com |
| 101 | Sachin | Clerk | 13-12-1910 | 13-06-2024 | HR | Sachin@gmail.com |

**DISTINCT Clause :-**

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

=> distinct clause eliminates duplicates from select statement output

SELECT DISTINCT COLNAME

SELECT DISTINCT COL1,COL2

SELECT DISTINCT \*

**Ex :-**

1 SELECT DISTINCT JOB FROM EMP

ANALYST

CLERK

MANAGER

PRESIDENT

SALESMAN

2 SELECT DISTINCT DEPTNO FROM EMP

10

20

30

3

EMP44

ENO ENAME SAL

1 A 5000

2 B 6000

3 C 7000

1 A 5000

2 B 6000

SELECT DISTINCT \* FROM EMP44

1 A 5000

2 B 6000

3 C 7000

Note : when Distinct is used with \* then it will select only unique records.

**TOP clause :-**

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

=> Used to display top N rows.

=> The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

**Syntax:**

SELECT TOP <n> colnames / \*

FROM tabname

[WHERE cond]

[ORDER BY --]

Ex :-

=> display first 5 rows from emp table?

SELECT TOP 5 \*

FROM EMP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** | **EMAIL** |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR | Sachin@gmail.com |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT | Virat@gmail.com |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES | David@gmail.com |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT | Anand@gmail.com |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES | Rahul@gmail.com |

=> display top 5 highest paid employees ?

SELECT TOP 5 \*

FROM EMP

ORDER BY SAL DESC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** | **EMAIL** |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES | Rahul@gmail.com |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT | Virat@gmail.com |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES | David@gmail.com |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT | Anand@gmail.com |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR | Sachin@gmail.com |

=> display top 5 employees based on experience?

SELECT TOP 5 \*

FROM EMP

ORDER BY HIREDATE ASC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Empid** | **Ename** | **Job** | **Sal** | **Hiredate** | **Dept** | **Email** |
| 104 | Anand | Clerk | 5000 | 20-04-2018 | IT | Anand@gmail.com |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES | Rahul@gmail.com |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES | David@gmail.com |
| 101 | Sachin | Clerk | 4000 | 13-06-2024 | HR | Sachin@gmail.com |
| 106 | Vijay | NULL | NULL | 27-07-2024 | IT | Vijay@gmail.com |

=> display top 3 max salaries?

SELECT DISTINCT TOP 3 SAL

FROM EMP

ORDER BY SAL DESC

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EMPID** | **ENAME** | **JOB** | **SAL** | **HIREDATE** | **DEPT** | **EMAIL** |
| 105 | Rahul | Manager | 10000 | 12-09-2019 | SALES | Rahul@gmail.com |
| 102 | Virat | Manager | 8000 | 27-07-2024 | IT | Virat@gmail.com |
| 103 | David | Analyst | 6000 | 15-08-2020 | SALES | David@gmail.com |

**SUMMARY :-**

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

WHERE => to select specific rows / to filter out rows based on conditions

ORDER BY => to sort table data ascending or descending

DISTINCT => to eliminate duplicates records or values

TOP => to display top n rows

Schema: A database schema outlines how key elements in a relational database, such as tables and records, are organized and connected with each other.

In short, it is overall structural design of the database.

<https://www.geeksforgeeks.org/create-schema-in-sql-server/>

<https://www.fivetran.com/learn/what-is-a-database-schema>

*SQL Server has some built-in schema, for example, dbo, guest, sys, and INFORMATION\_SCHEMA.*

*dbo is the default schema for a new database, owned by dbo user. While creating a new user with CREATE USER command, the user will take dbo as its default schema.*

Few schemas come with default tables like **INFORMATION\_SCHEMA** has default table as **Tables**. Default schema is DBO schema i.e. if we don’t mention schema name then it will search it in DBO schema.

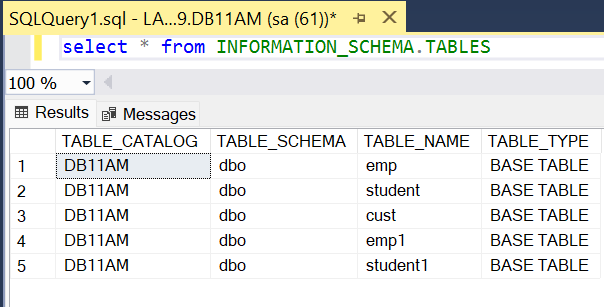
System created tables comes under the **SYS** and **INFORMATION\_SCHEMA** schemas. User created tables comes under the DBO ( Database Owner Schema)

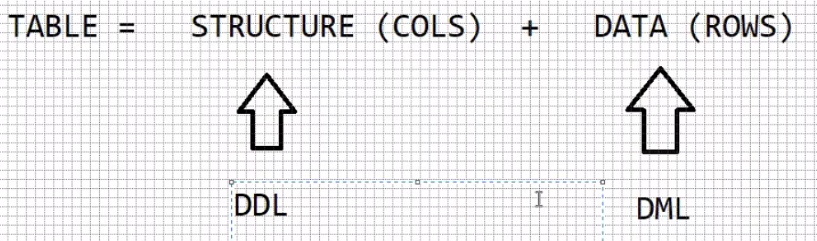
=> list of tables ?

SELECT \* FROM INFORMATION\_SCHEMA.TABLES

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

SCHEMA TABLE



****

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

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

INSERT

UPDATE

DELETE

MERGE

=> all DML commands acts on table data.

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

=> to stop auto commit execute the following command

SET IMPLICIT\_TRANSACTIONS ON

=> **“SET IMPLICIT\_TRANSACTIONS ON”** makes it auto commit off. i.e. we can manually save or rollback the changes.

=> to save the operation execute COMMIT

=> to cancel the operation, execute ROLLBACK

create table emp3(empno int,ename varchar(20),job varchar(20), mgr int, hiredate date, sal money, comm money, deptno int)

insert into emp3 values

(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20),

(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30),

(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30),

(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20),

(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30),

(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30),

(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10),

(7788,'SCOTT','ANALYST',7566,'1982-12-09',3000,NULL,20),

(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10),

(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30),

(7876,'ADAMS','CLERK',7788,'1983-01-12',1100,NULL,20),

(7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30),

(7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20),

(7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10)

**UPDATE command :-**

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

=> command used to modify table data.

=> we can update all rows or specific rows.

=> we can update single column or multiple columns.

**syntax :-**

UPDATE <tabname>

SET colname = value , colname = value , ----------

[WHERE cond]

**ex :-**

=> update all employees comm with 500?

UPDATE EMP SET COMM = 500

=> update employees comm with 500 for those who are not earning comm ?

UPDATE EMP SET COMM = 500 WHERE COMM IS NULL

=> update employees comm with null those who are earning comm ?

UPDATE EMP SET **COMM = NULL** WHERE **COMM IS NOT NULL**

**NULL assignment =**

**NULL comparison IS**

=> update employee sal with 1000 and comm with 500 whose empno = 7369 ?

UPDATE EMP SET SAL = 1000 , COMM = 500 WHERE EMPNO = 7369

=> increment sal by 20% and comm by 10% those working as salesman

and joined in 1981 year ?

UPDATE EMP

SET SAL = SAL + (SAL\*0.2) , COMM = COMM + (COMM\*0.1)

WHERE JOB='SALESMAN'

AND

HIREDATE LIKE '1981%'

20-JUN-24

=> increase price of samsung,redmi,realme mobiles by 10% ?

products

prodid pname price category brand

UPDATE PRODUCTS

SET PRICE = PRICE + (PRICE \* 0.1)

WHERE BRAND IN ('samsung','redmi','realme')

AND

CATEGORY = 'mobiles'

**DELETE command :-**

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

=> command used to delete row/rows from table based on a condition

**syntax**: - DELETE FROM <tabname> [WHERE cond]

=> delete all rows from emp table ?

DELETE FROM EMP

=> delete employees joined in 1980 year ?

DELETE FROM EMP WHERE HIREDATE LIKE '1980%'

=> delete employees working as salesman and manager belongs 30th dept ?

DELETE FROM EMP WHERE JOB IN ('SALESMAN','MANAGER')

AND

DEPTNO = 30

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

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

CREATE

ALTER

DROP

TRUNCATE

=> all DDL commands acts on table structure.

=> all DDL commands are auto committed.

=> to stop auto commit execute

**SET IMPLICIT\_TRANSACTIONS ON**

ALTER command :-

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

=> command used to modify table structure.

=> using alter command we can

1 add columns

2 drop columns

3 modify a column (If there is appropriate data of targeted datatype and size)

changing datatype

changing size

**Adding columns :-**

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

ALTER TABLE <tabname>

ADD colname datatype(size) , -----------

ex :-

=> add column gender to emp table ?

ALTER TABLE EMP

ADD GENDER CHAR(1)

after adding by default the new column is filled with NULL values, to insert

data into the new column use update command.

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

**Dropping column :-**

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

**ALTER TABLE <TABNAME>**

**DROP COLUMN COL1,COL2,-----**

**Limitations**

You can't delete a column that has a CHECK constraint. You must first delete the constraint.

You can't delete a column that has PRIMARY KEY or FOREIGN KEY constraints or other dependencies except when using the Table Designer in SSMS. When using Object Explorer in SSMS or Transact-SQL, you must first remove all dependencies on the column.

ex :-

=> drop column gender from emp ?

ALTER TABLE EMP

DROP COLUMN GENDER

**Modifying a column :-**

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

1 changing datatype

2 changing size

**ALTER TABLE <tabname>**

**ALTER COLUMN COLNAME DATATYPE(SIZE)**

Ex :-

=> modify empno datatype to int ?

ALTER TABLE EMP

ALTER COLUMN EMPNO INT

**ALTER TABLE EMP**

**ALTER COLUMN EMPNO TINYINT => ERROR**

=> modify column comm datatype to smallmoney ?

ALTER TABLE EMP

ALTER COLUMN COMM SMALLMONEY

=> 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.**

**DROP command :-**

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

=> command used to drop table from db.

=> drops structure along with data.

**Syntax :- DROP TABLE <TABNAME>**

Ex :- DROP TABLE EMP

**TRUNCATE command :-**

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

=> deletes all the data but keeps structure

=> will make empty the current table

=> releases memory allocated for table

**Syntax :- TRUNCATE TABLE <TABNAME>**

ex :- TRUNCATE TABLE EMP

**=> SQL server goes to memory and releases all the pages allocated for table**

**when pages are released, data stored in pages also deleted.**

21-jun-24

**DELETE VS TRUNCATE :-**

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

|  |  |  |
| --- | --- | --- |
| **Sr No** | **DELETE** | **TRUNCATE** |
| 1 | DML | DDL |
| 2 | We can delete all rows and specific rows. | We can delete only all rows but cannot delete specific rows |
| 3 | Where condition can be used with delete. | Where condition cannot be used with truncate. |
| 4 | Deletes records row-by-row | Deletes all rows at a time |
| 5 | Slower | Faster |
| 6 | Will not release memory | Releases memory |
| 7 | Will not reset identity | Will reset identity |

**SP\_RENAME** :- (sp => stored procedure)

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

=> used to change tablename and column name

SP\_RENAME 'oldname' , 'newname'

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

=> functions are used for data processing, analysis, cleansing etc

Types of functions: -

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

1. Date
2. Character
3. Numeric
4. Conversion
5. Special
6. Analytical
7. Aggregate

**Date Functions: -**

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

**GETDATE() :-**

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

=> returns date, time and also milliseconds.

Ex :-

SELECT GETDATE() => 2024-06-21 11:55:36.640

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

DATE TIME MS

**DATEPART() :-**

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

=> Used to extract part of the date

DATEPART(interval,date)

ex:-

SELECT DATEPART(YY,GETDATE()) => 2024

MM => 06

DD => 21

DW => 6 (1-7)

HH => 12

MI => minutes

SS => seconds

QQ => 2 (1-4)

jan-mar 1

apr-jun 2

jul-sep 3

oct-dec 4

DY => day of the year (1-366)

|  |  |
| --- | --- |
| select getdate()as [Full Date], | Full Date  2024-08-07 13:16:57.980 |
| DATEPART(YY,GETDATE()) as Year, | Year  2024 |
| DATEPART(YYYY,GETDATE()) as Year, | Year  2024 |
| DATEPART(MM,GETDATE()) as Month, | Month  8 |
| DATEPART(DD,GETDATE()) as [Day of Month], | Day of Month  7 |
| DATEPART(DW,GETDATE()) as [Day of Week], | Day of Week  4 |
| DATEPART(HH,GETDATE()) as [Hours of Day], | Hours of Day  13 |
| DATEPART(MI,GETDATE()) as [Minutes of Day], | Minutes of Day  16 |
| DATEPART(SS,GETDATE()) as [SECONDS of Minute], | SECONDS of Minute  57 |
| DATEPART(QQ,GETDATE()) as [Quarter of Year], | Quarter of Year  3 |
| DATEPART(DY,GETDATE()) as [Day of Year] | Day of Year  220 |

=> Show all employees

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **EMP NO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPT NO** | **GENDER** |
| 7369 | SMITH | CLERK | 7902 | 17-12-1980 | 1000 | 500 | 20 | NULL |
| 7499 | ALLEN | SALESMAN | 7698 | 20-02-1981 | 1600 | 300 | 30 | NULL |
| 7521 | WARD | SALESMAN | 7698 | 22-02-1981 | 1250 | 500 | 30 | NULL |
| 7566 | JONES | MANAGER | 7839 | 02-04-1981 | 2975 | NULL | 20 | NULL |
| 7654 | MARTIN | SALESMAN | 7698 | 28-09-1981 | 1250 | 1400 | 30 | NULL |
| 7698 | BLAKE | MANAGER | 7839 | 01-05-1981 | 2850 | NULL | 30 | NULL |
| 7782 | CLARK | MANAGER | 7839 | 09-06-1981 | 2450 | NULL | 10 | NULL |
| 7788 | SCOTT | ANALYST | 7566 | 09-12-1982 | 3000 | NULL | 20 | NULL |
| 7839 | KING | PRESIDENT | NULL | 17-11-1981 | 5000 | NULL | 10 | NULL |
| 7844 | TURNER | SALESMAN | 7698 | 08-09-1981 | 1500 | 0 | 30 | NULL |
| 7876 | ADAMS | CLERK | 7788 | 12-01-1983 | 1100 | NULL | 20 | NULL |
| 7900 | JAMES | CLERK | 7698 | 03-12-1981 | 950 | NULL | 30 | NULL |
| 7902 | FORD | ANALYST | 7566 | 03-12-1981 | 3000 | NULL | 20 | NULL |
| 7934 | MILLER | CLERK | 7782 | 23-01-1982 | 1300 | NULL | 10 | NULL |

=> employees joined in 1980,1983,1985 ?

SELECT \*

FROM EMP

WHERE DATEPART(YY,HIREDATE) IN (1980,1983,1985)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPTNO** | **GENDER** |
| 7369 | SMITH | CLERK | 7902 | 17-12-1980 | 1000 | 500 | 20 | NULL |
| 7876 | ADAMS | CLERK | 7788 | 12-01-1983 | 1100 | NULL | 20 | NULL |

=> employees joined in leap year ?

SELECT \*

FROM EMP

WHERE DATEPART(YY,HIREDATE) % 4 = 0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPTNO** | **GENDER** |
| 7369 | SMITH | CLERK | 7902 | 17-12-1980 | 1000 | 500 | 20 | NULL |

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

SELECT \*

FROM EMP

WHERE DATEPART(MM,HIREDATE) IN (1,4,12)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPTNO** |
| 7369 | SMITH | CLERK | 7902 | 17-12-1980 | 1000 | 500 | 20 |
| 7566 | JONES | MANAGER | 7839 | 02-04-1981 | 2975 | NULL | 20 |
| 7788 | SCOTT | ANALYST | 7566 | 09-12-1982 | 3000 | NULL | 20 |
| 7876 | ADAMS | CLERK | 7788 | 12-01-1983 | 1100 | NULL | 20 |
| 7900 | JAMES | CLERK | 7698 | 03-12-1981 | 950 | NULL | 30 |
| 7902 | FORD | ANALYST | 7566 | 03-12-1981 | 3000 | NULL | 20 |
| 7934 | MILLER | CLERK | 7782 | 23-01-1982 | 1300 | NULL | 10 |

=> employees joined on sunday ?

SELECT \*

FROM EMP

WHERE DATEPART(DW,HIREDATE) = 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **EMP NO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPT NO** |
| 7521 | WARD | SALESMAN | 7698 | 22-02-1981 | 1250 | 500 | 30 |

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

SELECT \*

FROM EMP

WHERE DATEPART(YY,HIREDATE) = 1981

AND

DATEPART(QQ,HIREDATE) = 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **EMP NO** | **ENAME** | **JOB** | **MGR** | **HIREDATE** | **SAL** | **COMM** | **DEPT NO** |
| 7566 | JONES | MANAGER | 7839 | 02-04-1981 | 2975 | NULL | 20 |
| 7698 | BLAKE | MANAGER | 7839 | 01-05-1981 | 2850 | NULL | 30 |
| 7782 | CLARK | MANAGER | 7839 | 09-06-1981 | 2450 | NULL | 10 |

**DATENAME() :-**

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

=> datename also returns part of the date

DATEPART DATENAME

MM 6 JUNE

DW 6 FRIDAY

|  |  |
| --- | --- |
| select  hiredate, | **HIREDATE**  1980-12-17  1981-02-20  1981-02-22 |
| DATENAME(YY,hiredate) AS YY, | **YY**  1980  1981  1981 |
| DATENAME(DY,hiredate) AS | **DY**  352  51  53 |
| DY,DATENAME(MM,hiredate) AS MM, | **MM**  December  February  February |
| DATENAME(DW,hiredate) AS DW, | **DW**  Wednesday  Friday  Sunday |
| DATENAME(QQ,hiredate) AS QQ from emp2 | **QQ**  4  1  1 |

=> display ENAME DAY ?

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

|  |  |
| --- | --- |
| **ENAME** | **DAY** |
| SMITH | Wednesday |
| ALLEN | Friday |
| WARD | Sunday |

=> display on which day india got independence ?

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

**Format() :-**

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

=> used to display dates in different formats

FORMAT(date,'format')

ex :-

SELECT FORMAT(getdate(), 'dd.MM.yyyy') => 21.06.2024

SELECT FORMAT(getdate(), 'MM/dd/yyyy') => 06/21/2024

SELECT FORMAT(getdate(), 'hh:mm:ss') => 12:36:51

SELECT FORMAT(getdate(), 'dd-MM-yyyy hh:mm:ss dddd') => 21-06-2024 12:38:27 Friday

|  |  |
| --- | --- |
| select getdate(), | 2024-08-07 16:02:03.697 |
| format(getdate(),'dd.MM.yyyy'), | 07.08.2024 |
| format(getdate(),'MM/dd/yyyy'), | 08/07/2024 |
| format(getdate(),'hh:mm:ss'), | 04:02:03 |
| format(getdate(),'dd-MM-yyyy hh:mm:ss dddd') | 07-08-2024 04:02:03 Wednesday |

**Scenario :-**

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

INSERT INTO EMP(EMPNO,ENAME,SAL,HIREDATE)

VALUES(999,'ABC',4000,GETDATE())

**List of employees joined today ?**

SELECT \* FROM EMP WHERE HIREDATE = GETDATE()

2024-06-21 = 2024-06-21 12:45:20.123

SELECT \*

FROM EMP

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

2024-06-21 = 2024-06-21

22-jun-24

**DATEDIFF() :-**

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

=> returns difference between two dates in years,months,days etc

DATEDIFF(interval,date1,date2)

ex :-

SELECT DATEDIFF(YY,'2023-06-22',GETDATE()) => 1

SELECT DATEDIFF(MM,'2023-06-22',GETDATE()) => 12

SELECT DATEDIFF(DD,'2023-06-22',GETDATE()) => 366

=> display ENAME EXPERIENCE in years ?

SELECT ENAME,DATEDIFF(YY,HIREDATE,GETDATE()) AS EXPR 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

|  |  |  |
| --- | --- | --- |
| **ENAME** | **YEARS** | **MONTHS** |
| SMITH | 43 | 8 |
| ALLEN | 43 | 6 |
| WARD | 43 | 6 |
| JONES | 43 | 4 |
| MARTIN | 42 | 11 |
| BLAKE | 43 | 3 |
| CLARK | 43 | 2 |
| SCOTT | 41 | 8 |
| KING | 42 | 9 |
| TURNER | 42 | 11 |
| ADAMS | 41 | 7 |
| JAMES | 42 | 8 |
| FORD | 42 | 8 |
| MILLER | 42 | 7 |

**DATEADD() :-**

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

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

DATEADD(interval,int,date)

ex :-

SELECT DATEADD(yy,1,getdate()) => 2025-06-22

SELECT DATEADD(yy,-1,getdate()) => 2023-06-22

SELECT DATEADD(mm,2,getdate()) => 2024-08-22

**Note : Decimal values not supported as it is int**

=>

**GOLD\_RATES**

**DATEID RATE**

2020-01-01 ?

2020-01-02 ?

2024-06-22 ?

=> display today's gold rate ?

SELECT \*

FROM GOLD\_RATES

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

2024-06-22

=> display yesterday's gold rate ?

SELECT \*

FROM GOLD\_RATES

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

**2024-08-07**

=> display last month same day gold rate ?

SELECT \*

FROM GOLD\_RATES

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

2024-07-08

=> display last year same day gold rate ?

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

=> display last 1 month gold rate ?

SELECT \*

FROM GOLD\_RATES

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

EOMONTH() :-

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

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

EOMONTH(date,int) -

Ex :-

SELECT EOMONTH(GETDATE(),0) => 2024-06-30

SELECT EOMONTH(GETDATE(),-1) => 2024-05-31

SELECT EOMONTH(GETDATE(), 1) => 2024-07-31

=> display current month 1st day ?

=> display next month 1st day ?

=> display current year 1st day ?

=> display next year 1st day ?

character functions :-

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

UPPER() :-

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

=> converts string to uppercase

UPPER(arg)

ex :-

SELECT UPPER('hello') => HELLO

LOWER() :-

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

=> converts string to lowercase

LOWER(arg)

ex :-

SELECT LOWER('HELLO') => hello

=> display EMPNO ENAME SAL ?

display names in lowercase ?

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

=> convert names to lowercase in table ?

UPDATE EMP SET ENAME = LOWER(ENAME)

LEN() :-

-----------

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

LEN(arg)

ex :-

SELECT LEN('HELLO WELCOME') => 13

=> employees name contains more than 5 chars ?

SELECT \*

FROM EMP

WHERE LENGTH(ENAME) > 5

LEFT() :-

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

=> returns chars starting from left side

LEFT(string,no of chars)

ex :-

SELECT LEFT('HELLO WELCOME',5) => HELLO

=> employees name starts with 's' ?

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

24-jun-24

RIGHT() :-

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

=> returns chars starting from right side

RIGHT(string,length)

ex :-

SELECT RIGHT('HELLO WELCOME',7) => WELCOME

=> employees name ends with 's' ?

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

=> employees name starts and ends with same char ?

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

OR

ENAME LIKE 'b%b'

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

=> update employee names with initial letter capital ?

UPDATE EMP SET ENAME = UPPER(LEFT(ENAME,1)) +

RIGHT(ename,LEN(ename)-1)

=> display EMPNO ENAME EMAILID ? generate emailid as follows

EMPNO ENAME EMAILID

7369 smith smi736@tcs.com

7499 allen all749@tcs.com

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(20)

step 2 :- update the column with emailids

UPDATE EMP

SET EMAILID = LEFT(ENAME,3) + LEFT(EMPNO,3) + '@tcs.com'

SUBSTRING() :-

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

=> returns chars starting from specific position

SUBSTRING(string,start,no of chars)

ex :-

SELECT SUBSTRING('HELLO WELCOME',1,5) => HELLO

SELECT SUBSTRING('HELLO WELCOME',7,4) => WELC

SELECT SUBSTRING('HELLO WELCOME',10,3) => COM

CHARINDEX() :-

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

=> returns position of a char in a string

charindex(char,string,[start])

ex :-

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

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

SELECT CHARINDEX('e','hello welcome',10) => 13

=> display CID FNAME LNAME ?

CUST

CID CNAME

10 sachin tendulkar

11 virat kohli

FNAME = SUBSTRING(CNAME,1,CHARINDEX(' ',CNAME)-1)

LNAME = SUBSTRING(CNAME,CHARINDEX(' ',CNAME)+1,LEN(CNAME))

SELECT CID,

SUBSTRING(CNAME,1,CHARINDEX(' ',CNAME)-1) AS FNAME,

SUBSTRING(CNAME,CHARINDEX(' ',CNAME)+1,LEN(CNAME)) AS LNAME

FROM CUST

REPLICATE() :-

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

=> repeats given char for number of times

replicate(char,length)

ex :-

SELECT REPLICATE('\*',10) => \*\*\*\*\*\*\*\*\*\*

=> display ENAME SAL ? display salaries in '\*' ?

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

=> your a/c no XXXX9572 debited ?

ACCOUNTS

ACCNO BAL

123456789572 10000

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

REPLACE() :-

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

=> used to replace one string with another string

REPLACE(str1,str2,str3)

ex :-

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)

ex :-

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

e => a

l => b

o => c

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

cipher text.

SELECT ENAME, TRANSLATE(SAL,'0123456789.','$Kb\*T@#&^%!') AS SAL FROM EMP

jones 2975.00 b%&@!$$

=> remove all special chars from '@#HE&^LL!%O$\*' ?

SELECT REPLACE(TRANSLATE( '@#HE&^LL!%O$\*','@#&^!%$\*','\*\*\*\*\*\*\*\*'),'\*','')

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

\*\*HE\*\*LL\*\*O\*\*

STUFF() :-

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

=> used to replace one string with another sting based on start and length

STUFF(str1,start,length,str2)

Ex :-

SELECT STUFF('HELLO WELCOME',7,7,'WORLD') => HELLO WORLD

25-jun-24

Numeric Functions :-

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

rounding numbers :-

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

ROUND

FLOOR

CEILING

ROUND :-

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

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

ROUND(number,decimal places)

ex :-

SELECT ROUND(38.45896,0) => 38

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

number < avg => rounded to lowest

number >= avg => rounded to highest

SELECT ROUND(38.55896,0) => 39

SELECT ROUND(38.45896,2) => 38.46

SELECT ROUND(38.45896,3) => 38.459

SELECT ROUND(38.45896,4) => 38.459

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

=> round sal to hundreds in table ?

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

FLOOR() :-

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

=> rounds number always to lowest

FLOOR(number)

Ex :-

SELECT FLOOR(3.9) => 3

CEILING() :-

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

=> rounds number always to highest

CEILING(number)

Ex :-

SELECT CEILING(3.1) => 4

conversion functions :-

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

=> functions used to convert one type to another type

1 CAST

2 CONVERT

CAST :-

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

CAST(expr as target-type)

ex :-

SELECT CAST(10.5 AS INT) => 10

=> display smith earns 800 ?

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

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

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

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

FROM emp

CONVERT() :-

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

CONVERT(target-type , expr)

ex :-

SELECT CONVERT(INT,10.5) => 10

=> difference between CAST & CONVERT ?

1 using convert we can display dates in different formats but not possible

using cast

2 using convert we can display numbers in different formats but not

possible using cast

Displaying dates in different formats :-

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

=> to display dates in different formats first convert date to char type.

CONVERT(VARCHAR,DATE,STYLE-NUMBER)

Ex :-

SELECT CONVERT(VARCHAR,GETDATE(),101) => 06/25/2024

102 => 2024.06.25

110 => 06-25-2024

114 => 12:21:32:900

120 =>

121 =>

126

127

=> display ENAME HIREDATE in yyyymmdd ?

SELECT ENAME,CONVERT(VARCHAR,HIREDATE,112) AS HIREDATE

FROM EMP

Money styles :-

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

CONVERT(VARCHAR,MONEY,STYLE-NUMBER)

ex :-

style-number

0

1 => displays number with thousand separator

2

=> display ENAME SAL ?

display salaries with thousand separator ?

SELECT ENAME,

CONVERT(VARCHAR,SAL,1) AS SAL

FROM EMP

special functions :-

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

ISNULL() :-

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

=> used to convert null values

ISNULL(arg1,arg2)

if arg1 = null returns arg2

if arg1 <> null returns arg1 only

ex :-

SELECT ISNULL(100,200) => 100

SELECT ISNULL(NULL,200) => 200

=> display ENAME SAL BONUS TOTSAL ?

TOTSAL = SAL + BONUS

SELECT ENAME,SAL,BONUS,SAL+ISNULL(BONUS,0) AS TOTSAL

FROM EMP

=> display ENAME SAL BONUS ?

if bonus = NULL display NO COMM ?

SELECT ENAME,SAL,

ISNULL(CAST(BONUS AS VARCHAR),'NO COMM') AS COMM

FROM EMP

26-JUN-24

**Analytical Functions :-**

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

rank

dense\_rank

row\_number

lag

lead

rank & dense\_rank :-

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

=> both functions are used to find ranks

=> ranks are always based on some field

=> for rank functions data must be sorted

rank() over (order by colname asc/desc)

dense\_rank() over (order by colname asc/desc)

ex :-

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

get 1st rank ?

SELECT ENAME,SAL,

RANK() OVER (ORDER BY SAL DESC) AS RNK

FROM EMP

SELECT 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 functions ranks may not be in sequence but in dense\_rank

ranks are always in sequence.

SAL RNK DRNK

5000 1 1

4000 2 2

3000 3 3

3000 3 3

3000 3 3

2000 6 4

2000 6 4

1000 8 5

=> find ranks of the employees based on sal , if salaries are same then

ranking should be based on hiredate ?

SELECT ENAME,HIREDATE,SAL, DENSE\_RANK() OVER (ORDER BY SAL DESC,HIREDATE ASC) AS RNK FROM EMP

PARTITION BY clause :-

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

=> partition by clause is used to divide the table based on one or more fields.

=> used to find ranks with in group , for ex to find ranks with in dept

first we need to divide the table dept wise using partition by and

apply rank functions on each partition.

SELECT ENAME,SAL,DEPTNO,DENSE\_RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) AS RNK FROM EMP

ROW\_NUMBER() :-

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

=> returns record numbers

=> it is is also based on some field

=> for row\_number also data must be sorted

ROW\_NUMBER() OVER (ORDER BY COLNAME ASC/DESC)

ex :- row number based on empno

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

LAG & LEAD :-

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

LAG(colname,int) OVER (ORDER BY ) => returns previous value

LEAD(colname,int) OVER (-----) => returns next value

EX :-

SELECT EMPNO,ENAME,SAL, LAG(SAL,1) OVER (ORDER BY EMPNO ASC) AS PREV\_SAL FROM EMP

7369 smith 1000.00 NULL

7499 allen 1600.00 1000.00

7521 ward 1300.00 1600.00

7566 jones 3000.00 1300.00

=>

GOLD\_RATES

DATEID RATE

2024-06-01 6800

2024-06-02 6850

2024-06-03 6700

2024-06-26 6600

display DATEID RATE CHANGE

2024-06-01 6800 NULL

2024-06-02 6850 50

03 6700 -150

SELECT DATEID,RATE, RATE - LAG(RATE,1) OVER (ORDER BY DATEID ASC) AS CHANGE FROM GOLD\_RATES

=> display ENAME HIREDATE DAYS ?

SELECT EMPNO,ENAME,HIREDATE, DATEDIFF(DD,LAG(HIREDATE,1) OVER (ORDER BY HIREDATE ASC), HIREDATE) AS DAYS FROM EMP

27-jun-24

Aggregate Functions :-

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

=> these functions process multiple rows and returns one value

MAX

MIN

SUM

AVG

COUNT

COUNT(\*)

MAX() :-

-----------

=> returns maximum value

MAX(arg)

ex :-

SELECT MAX(SAL) FROM EMP => 5000

SELECT MAX(HIREDATE) FROM EMP => 1983-01-12

SELECT MAX(ENAME) FROM EMP => ward

MIN() :-

-----------

=> returns minimum value

MIN(arg)

Ex :-

SELECT MIN(SAL) FROM EMP => 1000.00

SUM() :-

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

=> returns total

SUM(arg)

Ex :-

SELECT SUM(SAL) FROM EMP => 29500

=> round total sal to thousands ?

SELECT ROUND(SUM(SAL),-3) FROM EMP => 30000

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

=> after rounding display total sal with thousand seperator ?

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

=> calculate total sal including comm ?

SELECT SUM(SAL+COMM) FROM EMP => 9700

SELECT SUM(SAL+ISNULL(COMM,0)) FROM EMP => 32500

AVG() :-

=> returns average value

AVG(arg)

Ex :-

SELECT AVG(SAL) FROM EMP => 2107.1428

=> round avg(sal) to highest ?

SELECT CEILING(AVG(SAL)) FROM EMP => 2108

NOTE :-

=> sum,avg functions cannot be applied on char,date columns

COUNT() :-

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

=> returns no of values present in a column

COUNT(arg)

Ex :-

SELECT COUNT(EMPNO) FROM EMP => 14

SELECT COUNT(COMM) FROM EMP => 5 => nulls are not counted

COUNT(\*) :-

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

=> returns no of rows in a table

Ex :-

SELECT COUNT(\*) FROM EMP => 14

T1

F1

10

NULL

20

NULL

30

COUNT(F1) => 3

COUNT(\*) => 5

=> no of employees joined in 1981 year ?

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

AND

DATEPART(QQ,HIREDATE) = 2

NOTE :-

SELECT COL1,COL2 FROM TABNAME

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

SELECT ENAME,COUNT(\*) FROM EMP => ERROR

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

14 1

SELECT ENAME,ROUND(SAL,-2) AS SAL FROM EMP => EXECUTED

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

14 14

SELECT MIN(SAL),MAX(SAL) FROM EMP => EXECUTED

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

1 1

Summary :-

DATE :- datepart,datename,dateadd,datediff,eomonth

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

NUMERIC :- round,ceiling,floor

CONVERSION :- cast,convert

SPECIAL :- isnull

ANALYTICAL :- rank,dense\_rank,row\_number,lag,lead

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

CASE statement :-

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

=> similar to switch case and used to implement IF-ELSE in SQL queries

=> using case we can return values based on condition

=> case statements are 2 types

1 simple case

2 searched case

simple case :-

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

=> use simple case when conditions based on "=" operator

CASE COLNAME / EXPR

WHEN VALUE1 THEN RETURN EXPR1

WHEN VALUE2 THEN RETURN EXPR2

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

ELSE RETURN EXPR

END

Ex :-

=> display ENAME JOB ?

IF JOB=CLERK DISPLAY WORKER

MANAGER BOSS

PRESIDENT BIG BOSS

ELSE EMPLOYEE

SELECT ENAME,

CASE JOB

WHEN 'CLERK' THEN 'WORKER'

WHEN 'MANAGER' THEN 'BOSS'

WHEN 'PRESIDENT' THEN 'BIG BOSS'

ELSE 'EMPLOYEE'

END AS JOB

FROM EMP

28-jun-24

=> increment 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

=> update gender 'M' TO 'F' AND 'F' TO 'M' ?

UPDATE EMP

SET GENDER = CASE GENDER

WHEN 'M' THEN 'F'

WHEN 'F' THEN 'M'

END

searched case :-

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

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

CASE

WHEN COND1 THEN RETURN EXPR1

WHEN COND2 THEN RETURN EXPR2

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

ELSE RETURN EXPR

END

=> display ENAME SAL SALRANGE ?

0-2000

2001-4000

>4000

SELECT ENAME,SAL,

CASE

WHEN SAL BETWEEN 0 AND 2000 THEN '0-2000'

WHEN SAL BETWEEN 2001 AND 4000 THEN '2001-4000'

ELSE '>4000'

END AS SALRANGE

FROM EMP

=> display SNO TOTAL AVG RESULT ?

STUDENT

SNO SNAME S1 S2 S3

1 A 80 90 70

2 B 30 60 50

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

==========================================================================

INTEGRITY CONSTRAINTS

=====================

=> Integrity constraints are rules to maintain data integrity i.e. data quality or data consistency.

=> Integrity constraints are used to prevent users from entering invalid data.

=> Integrity constraints are used to enfoce rules like min bal must be 1000.

Types of constraints :-

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

1 NOT NULL

2 UNIQUE

3 PRIMARY KEY

4 CHECK

5 FOREIGN KEY

6 DEFAULT

=> above constraints can be declared in two ways

1 column level

2 table level

column level :-

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

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

CREATE TABLE <TABNAME>

(

COLNAME DATATYPE(SIZE) CONSTRAINT ,

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

)

NOT NULL :-

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

=> not null constraint doesn't accept null values

Ex :-

CREATE TABLE EMP11

(

EMPID INT ,

ENAME VARCHAR(10) NOT NULL

)

INSERT INTO EMP11 VALUES(100,NULL) => ERROR

INSERT INTO EMP11 VALUES(101,'A')

UNIQUE :-

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

=> unique constraint doesn't accept duplicates

ex :-

CREATE TABLE CUST

(

CID INT ,

CNAME VARCHAR(10) NOT NULL,

EMAILID VARCHAR(20) UNIQUE

)

INSERT INTO CUST VALUES(10,'A','abc@gmail.com')

INSERT INTO CUST VALUES(11,'B','abc@gmail.com') => ERROR

INSERT INTO CUST VALUES(12,'C','')

INSERT INTO CUST VALUES(13,'D','') => ERROR

29-jun-24

PRIMARY KEY :-

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

=> primary key doesn't accept duplicates and nulls

=> it is combination of unique and not null

=> In tables one column must be there to uniquely identify and into that column duplicates

and nulls are not allowed, so declare that column with primary key.

ex :-

CREATE TABLE EMP12

(

EMPID INT PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL

)

INSERT INTO EMP12 VALUES(100,'A')

INSERT INTO EMP12 VALUES(100,'B') => ERROR

INSERT INTO EMP12 VALUES(NULL,'B') => ERROR

NOTE :-

=> 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 ,

CNAME VARCHAR(10) NOT NULL,

AADHARNO BIGINT UNIQUE NOT NULL ,

PANNO CHAR(10) UNIQUE NOT NULL

)

=> columns declared with UNIQUE NOT NULL are called secondary keys or alternate keys

Difference between UNIQUE & PRIMARY KEY ?

|  |  |
| --- | --- |
| **UNIQUE** | **PRIMARY KEY** |
| allows one null | doesn't allow nulls |
| a table can have multiple unique | a table can have only one primary key |
| SQL server creates non clustered index on unique column | SQL server creates clustered index on primary key column |

CHECK :-

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

=> use check constraint when rule based on condition.

CHECK(condition)

ex 1 :- sal must be min 3000

CREATE TABLE EMP13

(

EMPNO INT PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL,

SAL MONEY CHECK(SAL>=3000)

)

INSERT INTO EMP13 VALUES(100,'A',1000) => ERROR

INSERT INTO EMP13 VALUES(101,'B',5000)

INSERT INTO EMP13 VALUES(102,'C',NULL)

NOTE :- check constraint allows nulls

ex 2 :- gender must be 'M','F' ?

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

ex 3 :- amt must be multiple of 100

AMT MONEY CHECK(AMT%100=0)

ex 4 :- pwd must be min 6 chars

PWD VARCHAR(12) CHECK(LEN(PWD) >= 6)

ex 5 :- emailid must contain '@'

must end with '.com' or '.co' or '.in'

EMAILID VARCHAR(20) CHECK(EMAILID LIKE '%@%'

AND

(

EMAILID LIKE '%.com'

OR

EMAILID LIKE '%.co'

OR

EMAILID LIKE '%.in'

))

FOREIGN KEY :-

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

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

=> To establish relationship , take primary key / unique of one table and add it to

another table as foreign key and declare with references constraint.

ex :-

DEPT

DNO DNAME

10 HR

20 IT

30 SALES

EMP

EMPNO ENAME SAL DNO REFERENCES DEPT(DNO)

1 A 3000 10

2 B 5000 20

3 C 4000 90 => ERROR

4 D 3000 10

5 E 2000 NULL

=> values entered in fk column should match with values entered in pk/unique column

=> fk allows duplicates and nulls

=> after declaring fk a relationship is created between two tables called parent / child relationship

=> pk table is parent and fk table is child

CREATE TABLE DEPT55

(

DNO INT PRIMARY KEY,

DNAME VARCHAR(10) UNIQUE NOT NULL

)

INSERT INTO DEPT55 VALUES(10,'HR'),(20,'IT')

CREATE TABLE EMP55

(

EMPNO INT PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL,

SAL MONEY CHECK(SAL>=3000),

DNO INT REFERENCES DEPT55(DNO)

)

INSERT INTO EMP55 VALUES(1,'A',4000,10)

INSERT INTO EMP55 VALUES(2,'B',5000,90) => ERROR

INSERT INTO EMP55 VALUES(3,'C',3000,10)

INSERT INTO EMP55 VALUES(4,'D',3000,NULL)

01-jul-24

Relationship Types :-

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

1 one to one (1:1)

2 one to many (1:m)

3 many to one (m:1)

4 many to many (m:n)

=> by default SQL server creates one to many relationship between tables

1:1 relationship :-

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

=> To establish 1:1 relationship declare fk with unique constraint

ex :-

DEPT

DNO DNAME

10 HR

20 IT

MGR

MGRNO MNAME DNO REFERENCES DEPT(DNO) UNIQUE

1 A 10

2 B 20

many to many relationship :-

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

=> if relationship is many to many then create 3rd table and add primary keys

of both tables as foreign keys.

ex :-

student course

sid sname cid cname

1 A 10 .NET

2 B 11 SQL SERVER

registrations

sid cid dor fee

1 10 ? 2000

1 11 ? 2000

2 10 ? 2000

RELATIONAL MODEL :-

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

BANK

CODE NAME ADDR

---------

BRANCH

BRANCH\_ID NAME ADDR CODE (FK)

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

ACCOUNT

ACCNO ACTYPE BAL BRANCH\_ID (FK) CUSTID(FK)

-----------

LOAN

LOAN\_ID LOAN\_TYPE AMOUNT BRANCH\_ID (FK) CUSTID(FK)

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

CUSTOMER

CUSTID NAME ADDR PHONE

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

DEFAULT :-

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

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

ex :- HIREDATE DATE DEFAULT GETDATE()

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

ex :-

CREATE TABLE EMP44

(

EMPID INT PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL,

HIREDATE DATE DEFAULT GETDATE()

)

INSERT INTO EMP44(EMPID,ENAME) VALUES(100,'A')

INSERT INTO EMP44 VALUES(101,'B','2024-01-01')

INSERT INTO EMP44 VALUES(102,'C',NULL)

SELECT \* FROM EMP44

100 A 2024-07-01

101 B 2024-01-01

102 C NULL

=> create following tables with given rules ?

ACCOUNTS

ACCNO ACTYPE BAL

Rules :-

1 accno should not be duplicate & null

2 actype must be 'S' OR 'C'

3 bal must be min 1000

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

Rules :-

1 trid should not be duplicate & null

2 ttype must be 'w' or 'd'

3 tdate must be always current date

4 tamt must be multiple of 100

5 accno should match with accounts table accno

02-jul-24

TABLE LEVEL :-

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

=> table level constraints are declared after declaring all columns

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

CREATE TABLE <tabname>

(

colname datatype(size) ,

colname datatype(size),

------------------------------,

constraint

)

Declaring check constraint at table level :-

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

products

prodid pname price mfd\_dt exp\_dt

100 A 50 2024-06-01 2024-01-01 => INVALID

Rule :- exp\_dt > mfd\_dt

CREATE TABLE PRODUCTS

(

PRODID INT PRIMARY KEY,

PNAME VARCHAR(10),

PRICE SMALLMONEY,

MFD\_DT DATE ,

EXP\_DT DATE ,

CHECK(EXP\_DT > MFD\_DT)

)

INSERT INTO PRODUCTS VALUES(100,'A',50,GETDATE(),'2024-01-01') => ERROR

INSERT INTO PRODUCTS VALUES(100,'A',50,'2024-01-01',GETDATE())

Composite primary key :-

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

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

primary key.

=> In some tables we may not be able uniquely identify using single column

and we need combination of columns to uniquely identify and that

combination should be declared primary key at table level.

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

ex :-

student course

sid sname cid cname

1 A 10 .NET

2 B 11 SQL SERVER

registrations

sid cid dor fee

1 10 ? ?

1 11 ?

2 10 ? ?

=> In the above table sid,cid combination uniquely identifies so declare this

combination as primary key at table level.

create table student

(

sid int primary key,

sname varchar(10) not null

)

insert into student values(1,'A') , (2,'B')

create table course

(

cid int primary key,

cname varchar(10) not null

)

insert into course values(10,'.net'),(11,'sql server')

create table registrations

(

sid int references student(sid) ,

cid int references course(cid),

dor date ,

fee money,

primary key(sid,cid)

)

insert into registrations values(1,10,getdate(),5000)

insert into registrations values(1,11,getdate(),5000)

insert into registrations values(2,10,getdate(),5000)

insert into registrations values(1,10,getdate(),5000) => ERROR

=> identify pk and write create table script ?

SALES

DATEID PRODID CUSTID QTY AMT

2024-07-01 100 10 1 2000

2024-07-01 100 11 1 2000

2024-07-01 101 10 1 1000

2024-07-02 100 10 1 2000

composite foreign key :-

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

=> if combination of columns declare foreign key then it is called composite

foreign key.

=> a composite foreign key refers composite primary key.

ex :-

registrations

sid cid dor fee

1 10 ? ?

1 11 ?

2 10 ? ?

certificates

certno doi sid cid

1000 ?? 1 10

1001 ? 1 11

1002 ? 2 11

=> In the above table sid,cid combination should match with registrations table

sid,cid combination , so declare this combination as fk at table level.

create table cetificates

(

certno int primary key,

doi date,

sid int ,

cid int ,

foreign key(sid,cid) references registrations(sid,cid)

)

insert into certificates values(1000,getdate(),2,11) => ERROR

which of the following constraint cannot be declated at table level ?

A UNIQUE

B CHECK

C NOT NULL

D PRIMARY KEY

E FOREIGN KEY

ANS :- C

Which statements are true regarding constraints ?

A a foreign key cannot contain NULL value F

B a column with UNIQUE constraint can contain NULL value T

C a constraint is enforced only for the INSERT operation on a table F

D all constraints can be defined at column level and table level. F

Which CREATE TABLE statement is valid?

A. CREATE TABLE ord\_details

(ord\_no INT PRIMARY KEY,

item\_no INT PRIMARY KEY,

ord\_date DATE NOT NULL);

B. CREATE TABLE ord\_details

(ord\_no INT UNIQUE, NOT NULL,

item\_no INT,

ord\_date DATE DEFAULT GETDATE() NOT NULL);

C. CREATE TABLE ord\_details

(ord\_no INT,

item\_no INT,

ord\_date DATE DEFAULT NOT NULL,

UNIQUE (ord\_no),

PRIMARY KEY (ord\_no));

D. CREATE TABLE ord\_details

(ord\_no INT,

item\_no INT,

ord\_date DATE DEFAULT GETDATE() NOT NULL,

PRIMARY KEY (ord\_no, item\_no));

3-JUL-24

Adding constraints to existing table :-

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

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

create table emp66

(

empno int,

ename varchar(10) ,

sal money,

emailid varchar(20),

dno int

)

Adding primary key :-

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

=> primary key cannot be added to nullable column , to add primary key

first change the column to not null.

add primary key to empno ?

STEP 1 :-

ALTER TABLE EMP66

ALTER COLUMN EMPNO INT NOT NULL

STEP 2 :-

ALTER TABLE EMP66

ADD PRIMARY KEY(EMPNO)

Adding check constraint :-

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

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

ALTER TABLE EMP66

ADD CHECK(SAL>=3000)

ALTER TABLE EMP

ADD CHECK(SAL>=3000) => ERROR => some employee salaries

are less than 3000

note :- while adding constraint SQL server also validates existing data

WITH NOCHECK :-

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

=> if check constraint added "WITH NOCHECK" then SQL server will not

validate existing data and it validates only new data.

ALTER TABLE EMP

WITH NOCHECK ADD CHECK(SAL>=3000)

Adding foreign key :-

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

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

ALTER TABLE EMP66

ADD FOREIGN KEY(DNO) REFERENCES DEPT(DEPTNO)

Adding unique :-

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

=> add unique constraint to emailid ?

ALTER TABLE EMP66

ADD UNIQUE(EMAILID)

Changing from NULL to NOT NULL :-

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

=> modify the column ename to not null ?

ALTER TABLE EMP66

ALTER COLUMN ENAME VARCHAR(10) NOT NULL

Droping constraints :-

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

ALTER TABLE <TABNAME>

DROP CONSTRAINT <NAME>

Ex :-

=> drop check constraint in emp66 table ?

ALTER TABLE EMP66

DROP CONSTRAINT CK\_\_emp66\_\_sal\_\_08B54D69

=> drop primary key in dept table ?

ALTER TABLE DEPT

DROP CONSTRAINT PK\_\_DEPT\_\_E0EB08D76A0CED27 => ERROR

DROP TABLE DEPT => ERROR

TRUNCATE TABLE DEPT => ERROR

NOTE :-

=> pk cannot be dropped if referenced by some fk

=> pk table cannot be dropped if referenced by some fk

=> pk table cannot be truncated if referenced by some fk

DELETE rules :-

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

1 ON DELETE NO ACTION (DEFAULT)

2 ON DELETE CASCADE

3 ON DELETE SET NULL

4 ON DELETE SET DEFAULT

=> these rules are declared with foreign key

=> these rule specifies how child rows are affected if parent row is deleted

ON DELETE NO ACTION :-

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

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

create table dept99

(

dno int primary key,

dname varchar(10)

)

insert into dept99 values(10,'hr'),(20,'it')

create table emp99

(

empno int primary key,

ename varchar(10),

dno int references dept99(dno)

)

insert into emp99 values(1,'A',10)

DELETE FROM DEPT99 WHERE DNO=10 => ERROR

scenario :-

ACCOUNTS

ACCNO ACTYPE BAL

100 S 10000

101 S 20000

LOANS

ID TYPE AMT ACCNO REFERENCES ACCOUNTS(ACCNO)

1 H 40 100

2 C 10 100

ON DELETE CASCADE :-

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

=> parent row is deleted along with child rows.

create table dept99

(

dno int primary key,

dname varchar(10)

)

insert into dept99 values(10,'hr'),(20,'it')

create table emp99

(

empno int primary key,

ename varchar(10),

dno int references dept99(dno) ON DELETE CASCADE

)

insert into emp99 values(1,'A',10)

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

SELECT \* FROM EMP99 => no rows

scenario :-

ACCOUNTS

ACCNO ACTYPE BAL

100 S 10000

101 S 20000

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO REFERENCES ACCOUNTS(ACCNO)

1 W ? 2000 100 ON DELETE CASCADE

2 D ? 3000 100

ON DELETE SET NULL :-

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

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

create table dept99

(

dno int primary key,

dname varchar(10)

)

insert into dept99 values(10,'hr'),(20,'it')

create table emp99

(

empno int primary key,

ename varchar(10),

dno int references dept99(dno) ON DELETE SET NULL

)

insert into emp99 values(1,'A',10)

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

SELECT \* FROM EMP99

1 A NULL

scenario :-

projects

projid name duration

100

101

emp

empno ename projid references projects(projid) on delete set null

1 100

2 101

ON DELETE SET DEFAULT :-

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

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

create table dept99

(

dno int primary key,

dname varchar(10)

)

insert into dept99 values(10,'hr'),(20,'it')

create table emp99

(

empno int primary key,

ename varchar(10),

dno int default 20

references dept99(dno) ON DELETE SET DEFAULT

)

insert into emp99 values(1,'A',10)

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

SELECT \* FROM EMP99

1 A 20

summary :-

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

importance of constraints

types of constraints

declaring constraints

column level

table level

adding constraints

droping constraints

delete rules

===============================================================

JOINS

=====

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

=> In db related data stored in multiple tables; to gather or to combine data stored in multiple tables we need to join those tables.

ex :-

-------

orders cust

ordid orddt deldt cid cid cname addr

1000 01/ 05/ 10 10 A HYD

1001 02/ 07/ 11 11 B HYD

1002 03/ 10/ 12 12 C HYD

OUTPUT :-

ordid orddt deldt cname addr

1000 01/ 05/ A HYD

Types of joins :-

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

1 inner / equi join

2 outer join

left join

right join

full join

3 non equi

4 self join

5 cross join / cartesian join

Inner / Equi join :-

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

=> To perform inner join between the tables there must be a common field.

=> name of the common field need not to be same

=> pk-fk relationship is not compulsory

=> inner join is performed on common field with same datatype.

SELECT columns

FROM tab1 INNER JOIN tab2

ON join condition

join condition :-

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

=> join condition specifies which record of table1 joined with which record of table2

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

table1.commonfield = table2.commonfield

ex :-

EMP DEPT

EMPNO ENAME SAL DEPTNO DEPTNO DNAME LOC

1 A 3000 10 10 ACCT NEW YORK

2 B 4000 20 20 RESEARCH

3 C 5000 30 30 SALES

4 D 3000 20 40 OPERATIONS

5 E 2000 NULL

=> display ENAME SAL DNAME LOC ?

SELECT ENAME,SAL,DNAME,LOC

FROM EMP INNER JOIN DEPT

ON EMP.DEPTNO = DEPT.DEPTNO

A 3000 ACCT NEW YORK

B 4000 RESEARCH ?

C 5000 SALES ?

D 3000 RESEARCH ?

note :-

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

table alias for two reasons

1 to avoid ambiguity

2 for faster execution

SELECT E.ENAME,E.SAL,

D.DEPTNO,D.DNAME,D.LOC

FROM EMP AS E INNER JOIN DEPT AS D

ON E.DEPTNO = D.DEPTNO

=> display ENAME DNAME working at NEW YORK loc ?

SELECT E.ENAME,D.DNAME

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 2 tables :-

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

=> if no of tables increases no of join conditions also increases

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

SELECT columns

FROM tab1 INNER JOIN tab2

ON join cond

INNER JOIN tab3

ON join cond

INNER JOIN tab4

ON join cond

05-jul-24

ex :-

EMP DEPT LOCATIONS COUNTRIES

empno deptno locid country\_id

ename dname city country\_name

sal locid state

deptno country\_id

=> display ENAME DNAME CITY STATE COUNTRY ?

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

emp dept locations countries

SELECT E.ENAME,

D.DNAME,

L.CITY,L.STATE,

C.COUNTRY\_NAME

FROM EMP 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 ,

to display unmatched records perform outer join.

ex :-

EMP DEPT

EMPNO ENAME SAL DEPTNO DEPTNO DNAME LOC

1 A 3000 10 10 ACCT NEW YORK

2 B 4000 20 20 RESEARCH

3 C 5000 30 30 SALES

4 D 3000 20 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

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

A ACCOUNTS

B RESEARCH

C SALES

D RESEARCH

E NULL => unmatched from emp

RIGHT JOIN :-

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

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

SELECT E.ENAME,D.DNAME

FROM EMP AS E RIGHT JOIN DEPT AS D

ON E.DEPTNO = D.DEPTNO

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

A ACCOUNTS

B RESEARCH

C SALES

D RESEARCH

NULL OPERATIONS => unmatched from dept

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

=> returns all rows from emp & dept tables

A ACCOUNTS

B RESEARCH

C SALES

D RESEARCH

E NULL => unmatched from emp

NULL OPERATIONS => unmatched from dept

Displaying only unmatched records :-

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

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

E NULL

NULL OPERATIONS

=>

T1 T2

F1 C1

1 1

2 2

1 1

2 2

NULL NULL

NULL NULL

=> no of rows return by each join ?

inner join => 8

left join => 10

right join => 10

full join => 12

=>

emp projects

empid ename sal projid projid pname client

1 100 100

2 101 101

3 null 102

=> display employee details with project details ?

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

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

NON EQUI JOIN :-

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

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

ex :-

EMP SALGRADE

EMPNO ENAME SAL GRADE LOSAL HISAL

1 A 3000 1 700 1000

2 B 1000 2 1001 2000

3 C 5000 3 2001 3000

4 D 2500 4 3001 4000

5 E 1500 5 4001 9999

=> display ENAME SAL GRADE ?

SELECT E.ENAME,E.SAL,S.GRADE

FROM EMP AS E JOIN SALGRADE AS S

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

A 3000 3

B 1000 1

C 5000 5

D 2500 3

E 1500 2

=> display grade 4 employee list ?

SELECT E.ENAME,E.SAL,S.GRADE

FROM EMP AS E JOIN SALGRADE AS S

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

WHERE S.GRADE = 4

=> display ENAME DNAME GRADE ?

SELECT E.ENAME,D.DNAME,S.GRADE

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

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

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

EMP SALGRADE

EMPNO ENAME DEPTNO SAL GRADE LOSAL HISAL

1 A 10 3000 1 700 1000

2 B 20 1000 2 1001 2000

3 C 30 5000 3 2001 3000

4 D 10 2000 4 3001 4000

5 4001 9999

OUTPUT :- DEPT

DEPTNO DNAME

1 A 10 3000 3 10 ACCT

2 B 20 1000 1 20 RESEARCH

3 C 30 5000 5 30 SALES

4 D 10 2000 2

ON E.DEPTNO = D.DEPTNO :-

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

A ACCT 3

B RESEARCH 1

C SALES 5

D ACCT 2

SELF JOIN :-

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

=> joining a table to itself is called self join or recursive join.

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

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

FROM EMP AS X JOIN EMP AS Y

EMP X EMP Y

EMPNO ENAME MGR EMPNO ENAME MGR

7369 smith 7902 7369 smith 7902

7499 allen 7698 7499 allen 7698

7698 blake 7839 7698 blake 7839

7839 king null 7839 king null

7902 ford 7566 7902 ford 7566

=> display ENAME MGRNAME ?

SELECT X.ENAME,Y.ENAME AS MANAGER

FROM EMP AS X JOIN EMP AS Y

ON X.MGR = Y.EMPNO

smith ford

allen blake

blake king

=> employees reporting to blake ?

SELECT X.ENAME,Y.ENAME AS MANAGER

FROM EMP AS X JOIN EMP AS Y

ON X.MGR = Y.EMPNO

WHERE Y.ENAME='blake'

=> blake's manager name ?

SELECT X.ENAME,Y.ENAME

FROM EMP AS X JOIN EMP AS Y

ON X.MGR = Y.EMPNO

WHERE X.ENAME='blake'

=> employees earning more than their managers ?

SELECT X.ENAME,X.SAL,

Y.ENAME AS MANAGER,Y.SAL AS MGRSAL

FROM EMP AS X JOIN EMP AS Y

ON X.MGR = Y.EMPNO

WHERE X.SAL > Y.SAL

=> employees who are senior to their managers ?

SELECT X.ENAME,X.HIREDATE,

Y.ENAME AS MGR,Y.HIREDATE AS MGRHIRE

FROM EMP AS X JOIN EMP AS Y

ON X.MGR = Y.EMPNO

WHERE X.HIREDATE < Y.HIREDATE

=>

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 TEAMS B

ID COUNTRY ID COUNTRY

1 IND 1 IND

2 AUS 2 AUS

3 ENG 3 ENG

SELECT A.COUNTRY + ' VS ' + B.COUNTRY

FROM TEAMS AS A JOIN TEAMS AS B

ON A.ID < B.ID

IND AUS

IND ENG

AUS ENG

08-jul-24

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 the records

of second table.

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

ex :-

SELECT E.ENAME,D.DNAME FROM EMP AS E CROSS JOIN DEPT AS D

==========================================================================

SET OPERATORS :-

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

1 UNION

2 UNION ALL

3 INTERSECT

4 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 set of rows return by two queries

syntax :-

SELECT STATEMENT 1

UNION / UNION ALL / INTERSECT / EXCEPT

SELECT STATEMENT 2

Rules :-

-----------

1 no of columns return by both queries must be same

2 corresponding columns datatype must be same

SELECT JOB FROM EMP WHERE DEPTNO = 20

CLERK

MANAGER

ANALYST

CLERK

ANALYST

SELECT JOB FROM EMP WHERE DEPTNO = 30

SALESMAN

SALESMAN

SALESMAN

MANAGER

SALESMAN

CLERK

UNION :-

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

=> union combines rows return by two queries.

=> duplicates are eliminated

=> result is sorted

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

UNION VS JOIN :-

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

UNION JOIN

1 combines rows combines columns

2 horiontal merge vertical merge

3 performed between performed between

two queries two tables

ex 1 :-

T1 T2

F1 C1

1 10

2 20

3 30

T1 UNION T2 :- T1 JOIN T2 :-

1 1 10

2 2 20

3 3 30

10

20

30

ex 2 :-

emp\_us

eno ename sal dno

dept

emp\_ind dno dname loc

eno ename sal dno

=> total employees list?

SELECT \* FROM EMP\_US

UNION

SELECT \* FROM EMP\_IND

=> employees working at US loc with dept details ?

SELECT E.\*,D.\*

FROM EMP\_US 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

UNION VS UNION ALL :-

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

UNION UNION ALL

1 eliminates duplicates doesn't eliminate duplicates

2 sorts result result is not sorted

3 slower faster

INTERSECT :-

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

=> returns common values from the output of two queries

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

=>

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

9-JUL-24

GROUP BY clause :-

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

=> group by clause is used to group rows based on one or more columns to calculate min,max,sum,avg,count for each group. For ex to calculate dept wise total salary, first group the rows based on dept wise and apply sum function on each group.

emp

empno ename sal deptno

1 A 3000 10

2 B 6000 20 GROUP BY 10 8000

3 C 4000 30 ===============> 20 9000

4 D 5000 10 30 4000

5 E 3000 20

detailed data summarized data

=> group by clause can be used to convert detailed data into summarized data

which is useful for analysis.

syntax :-

SELECT columns

FROM tabname

[WHERE cond]

GROUP BY colname

[HAVING cond]

[ORDER BY col ASC/DESC]

Execution :-

FROM

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

Examples :-

=> display dept wise total salary ?

select deptno,sum(sal) as totsal

from emp

group by deptno

from emp :-

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

empno ename sal deptno

1 A 3000 10

2 B 6000 20

3 C 4000 30

4 D 5000 10

5 E 3000 20

group by deptno :-

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

10 1 A 3000

4 D 5000

20 2 B 6000

5 E 3000

30 3 C 4000

select deptno,sum(sal) as totsal :-

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

10 8000

20 9000

30 4000

=> display job wise no of employees ?

SELECT JOB,COUNT(\*) AS CNT

FROM EMP

GROUP BY JOB

ANALYST 2

CLERK 4

MANAGER 3

PRESIDENT 1

SALESMAN 4

=> display year wise no of employees joined ?

SELECT DATEPART(YY,HIREDATE) AS YEAR,COUNT(\*) AS CNT

FROM EMP

GROUP BY DATEPART(YY,HIREDATE)

1980 1

1981 10

1982 2

1983 1

=> day wise no of employees joined ?

SELECT DATENAME(DW,HIREDATE) AS DAY,COUNT(\*) AS CNT

FROM EMP

GROUP BY DATENAME(DW,HIREDATE)

Friday 2

Monday 1

Saturday 1

Sunday 1

Thursday 4

Tuesday 3

Wednesday 2

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

SELECT DATENAME(MM,HIREDATE) AS MONTH,COUNT(\*) AS CNT

FROM EMP

WHERE DATEPART(YY,HIREDATE)=1981

GROUP BY DATENAME(MM,HIREDATE)

=> find the departments having more than 3 employees ?

SELECT DEPTNO,COUNT(\*)

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

20 5

30 6

WHERE VS HAVING :-

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

WHERE HAVING

1 selects specific rows selects specific groups

2 conditions applied before conditions applied after

group by group by

3 use where clause if use having clause

cond doesn't contain if cond contains

aggregate function aggregate function

=> display dept wise no of employees where deptno = 10,20 and no of

employees > 3 ?

SELECT DEPTNO,COUNT(\*) AS CNT

FROM EMP

WHERE DEPTNO IN (10,20)

GROUP BY DEPTNO

HAVING COUNT(\*) > 3

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

PERSONS

AADHARNO NAME AGE GENDER ADDR CITY STATE

147+ CR

SELECT STATE,COUNT(\*) AS POPULATION

FROM PERSONS

WHERE STATE IN ('AP','TG','KA','TN','KL')

GROUP BY STATE

HAVING COUNT(\*) > 50000000

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

SELECT DEPTNO,JOB,SUM(SAL) AS TOTSAL

FROM EMP

GROUP BY DEPTNO,JOB

ORDER BY DEPTNO ASC

10 CLERK 1300

MANAGER 2450

PRESIDENT 5000

20 ANALYST 6000

CLERK 1900

MANAGER 2975

30 CLERK 950

MANAGER 2850

SALESMAN 5600

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

SELECT DATEPART(YY,HIREDATE) AS YEAR,

DATEPART(QQ,HIREDATE) AS QRT,COUNT(\*) AS CNT

FROM EMP

GROUP BY DATEPART(YY,HIREDATE) , DATEPART(QQ,HIREDATE)

ORDER BY DATEPART(YY,HIREDATE)

GROUP BY YEAR,QRT => INVALID

ORDER BY YEAR => VALID

NOTE :-

1 column alias cannot be used in where clause because where

is executed before select

2 column alias can be used in order by because order by is executed after select

10-JUL-24

ROLLUP & CUBE :-

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

=> both functions are used to display subtotals and grand total

GROUP BY ROLLUP(COL1,COL2,--)

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

ROLLUP :-

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

=> rollup displays subtotals for each group and also displays grand total.

SELECT DEPTNO,JOB,SUM(SAL) AS TOTSAL

FROM EMP

GROUP BY ROLLUP(DEPTNO,JOB)

ORDER BY DEPTNO ASC

NULL NULL 29025.00 => grand total

10 CLERK 1300.00

10 MANAGER 2450.00

10 PRESIDENT 5000.00

10 NULL 8750.00 => dept subtotal

CUBE :-

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

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

SELECT DEPTNO,JOB,SUM(SAL) AS TOTSAL

FROM EMP

GROUP BY CUBE(DEPTNO,JOB)

ORDER BY DEPTNO ASC

NULL NULL 29025.00 => grand total

NULL ANALYST 6000.00 => job subtotal

NULL CLERK 4150.00 => job subtotal

NULL SALESMAN 5600.00 => job subtotal

10 NULL 8750.00 => dept subtotal

10 CLERK 1300.00

10 MANAGER 2450.00

10 PRESIDENT 5000.00

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

SALES

DATEID PRODID CUSTID QTY AMT

2020-01-10 100 10 1 2000

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

PERSONS

AADHARNO NAME AGE GENDER ADDR CITY STATE

GROUPING\_ID() :-

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

=> this function accepts group by columns and returns subtotal belongs to which group by columns

ex :- GROUPING\_ID(deptno,job) =>

1 => subtotal belongs to 1st group by column i.e. deptno

2 => subtotal belongs to 2nd group by column i.e. job

3 => grand total

SELECT DEPTNO,JOB,SUM(SAL) AS TOTSAL,

CASE GROUPING\_ID(DEPTNO,JOB)

WHEN 1 THEN 'Dept subtotal'

WHEN 2 THEN 'Job subtotal'

WHEN 3 THEN 'Grand total'

END AS SUBTOTALS

FROM EMP

GROUP BY CUBE(DEPTNO,JOB)

ORDER BY DEPTNO ASC ,JOB ASC

===========================================================================

SUB-QUERIES / NESTED QUERIES :-

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

=> a query in another query is called sub-query or nested query

=> one query is called inner / child / sub-query

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

=> first SQL server executes inner query and next it executes outer query

=> inner query output is input to outer query

=> use sub-queries when where cond based on uknown value

Types of sub-queries :-

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

1 single row sub-queries

2 multi row sub-queries

3 co-related sub-queries

4 derived tables & CTEs

5 scalar sub-queries

single row sub-queries :-

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

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

SELECT columns

FROM tabname

WHERE colname OP (SELECT STATEMENT)

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

ex :-

=> employees earning more than blake ?

SELECT \*

FROM EMP

WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME='blake')

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

2850

=> employees who are senior to king ?

SELECT \*

FROM EMP

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

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

1981-11-17

=> name of the employee earnng max salary ?

SELECT ENAME ,MAX(SAL) FROM EMP => ERROR

14 1

SELECT COL1,COL2 FROM TABNAME

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

SELECT ENAME FROM EMP WHERE SAL = MAX(SAL) => aggregate functions are not

allowed in where clause

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)

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

1980-12-17

=> find the name having max no of chars ?

SELECT ENAME

FROM EMP

WHERE LEN(ENAME) = (SELECT MAX(LEN(ENAME)) FROM EMP)

=> display 2nd max salary ?

SELECT MAX(SAL)

FROM EMP

WHERE SAL <> (SELECT MAX(SAL) FROM EMP) => 3000

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

SELECT ENAME

FROM EMP

WHERE SAL = ( SELECT MAX(SAL)

FROM EMP

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

11-jul-24

=> increment sal by 10% having max experience ?

UPDATE EMP

SET SAL = SAL+(SAL\*0.1)

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

=> swap employee salaries whose empno = 7369,7499 ?

before swap after swap

7369 880 7369 1600

7499 1600 7499 880

UPDATE EMP

SET SAL = CASE EMPNO

WHEN 7369 THEN (SELECT SAL FROM EMP WHERE EMPNO=7499)

WHEN 7499 THEN (SELECT SAL FROM EMP WHERE EMPNO=7369)

END

WHERE EMPNO IN (7369,7499)

=> display employee names working at NEW YORK loc ?

join :-

SELECT E.ENAME

FROM EMP AS E INNER JOIN DEPT AS D

ON E.DEPTNO = D.DEPTNO

WHERE D.LOC = 'NEW YORK'

sub-query :-

SELECT ENAME

FROM EMP

WHERE DEPTNO = ( SELECT DEPTNO FROM DEPT WHERE LOC ='NEW YORK' )

=> display ENAME DNAME working at NEW YORK loc ?

join :-

SELECT E.ENAME,D.DNAME

FROM EMP AS E INNER JOIN DEPT AS D

ON E.DEPTNO = D.DEPTNO

WHERE D.LOC = 'NEW YORK'

sub-query :-

not possible

difference between join and sub-query ?

=> to display data from one table and condition based on another table then use

sub-query or join

=> to display data from two tables then use join operation

multi-row sub-queries :-

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

=> if inner query returns more than one value then it is called multi row sub-query

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

=> employee names working at NEW YORK,CHICAGO locations ?

SELECT ENAME

FROM EMP

WHERE DEPTNO IN (SELECT DEPTNO

FROM DEPT

WHERE LOC IN ('NEW YORK','CHICAGO'))

ANY , ALL operators :-

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

=> use ANY,ALL for > < comparision with multiple values

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

IF X = 800 FALSE IF X = 800 TRUE

1500 TRUE 1500 TRUE

4500 TRUE 4500 FALSE

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

IF X = 800 FALSE IF X = 800 TRUE

1500 FALSE 1500 FALSE

4500 TRUE 4500 FALSE

=> employees earning more than all managers ?

SELECT \*

FROM EMP

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

2975.00

2850.00

2450.00

=> employees earning more than atleast one manager ?

SELECT \*

FROM EMP

WHERE SAL > ANY(SELECT SAL FROM EMP WHERE JOB='MANAGER')

co-related sub-queries :-

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

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

=> execution starts from outer query and inner queyry is executed no of times

depends on no of rows return by outer query.

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

ex :-

EMP

EMPNO ENAME SAL DEPTNO

1 A 3000 10

2 B 6000 20

3 C 4000 30

4 D 5000 10

5 E 3000 20

=> employees earning more than avg sal ?

SELECT \*

FROM EMP

WHERE SAL > (SELECT AVG(SAL) FROM EMP)

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

4200

=> 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 3000 10 3000 > (4000) FALSE

2 B 6000 20 6000 > (4500) TRUE

3 C 4000 30 4000 > (4000) FALSE

4 D 5000 10 5000 > (4000) TRUE

5 E 3000 20 3000 > (4500) FALSE

12-JUL-24

=> employees earning max sal in their dept ?

SELECT \*

FROM EMP AS X

WHERE SAL = (SELECT MAX(SAL)

FROM EMP

WHERE DEPTNO = X.DEPTNO)

1 A 3000 10 3000 = (5000) FALSE

2 B 6000 20 6000 = (6000) TRUE

3 C 4000 30 4000 = (4000) TRUE

4 D 5000 10 5000 = (5000) TRUE

5 E 3000 20 3000 = (6000) FALSE

=> 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 A.SAL DESC

EMP A EMP B

SAL SAL

5000 5000 3 > (0) TRUE

4000 4000 3 > (1) TRUE

3000 3000 3 > (2) TRUE

2000 2000 3 > (3) FALSE

4000 4000 3 > (1) TRUE

=> display 5th max sal ?

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)

ORDER BY A.SAL DESC

DERIVED TABLES :-

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

=> sub-queries in FROM clause are called derived tables

SELECT columns

FROM (SELECT STATEMENT) AS <ALIAS>

WHERE COND

=> sub-query output acts like a table for outer query

=> derived tables are used in following scenarios

1 to control order of execution

2 to use result of one operation in another opration

3 to join two query outputs

controlling order of execution :-

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

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

FROM

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

=> use derived table to control this order of execution

SELECT SELECT

FROM ==============> FROM (SELECT

WHERE FROM

ORDER BY ORDER BY )

WHERE

Example 1 :-

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

rank ?

SELECT EMPNO,ENAME,SAL,

DENSE\_RANK() OVER (ORDER BY SAL DESC) AS RNK

FROM EMP

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

SELECT EMPNO,ENAME,SAL,

DENSE\_RANK() OVER (ORDER BY SAL DESC) AS RNK

FROM EMP

WHERE RNK<=5 => ERROR

=> column alias cannot be used in where clause because where clause

is executed before select. To overcome this use INLINE views.

SELECT \*

FROM ( SELECT EMPNO,ENAME,SAL,

DENSE\_RANK() OVER (ORDER BY SAL DESC) AS RNK

FROM EMP) AS E

WHERE RNK <= 5

=> 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

=> 5th max sal ?

WHERE RNK = 5

=> 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 = 5

WHERE RNO IN (5,10,14)

WHERE RNO BETWEEN 5 AND 10

WHERE RNO%2 = 0

=> display last 3 rows ?

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)

13-JUL-24

=> delete first 3 rows from emp ?

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 tables outer query cannot be DML and it must be always SELECT.

To overcome this problem use CTEs

CTE :- (common table expression)

---------

=> using CTE we can give name to the query output and we can use that name

in another query like INSERT / UPDATE / DELETE

=> In CTEs outer query can be INSERT / UPDATE / DELETE / SELECT.

=> CTEs are used to simplify complex queries.

WITH <cte-name1>

AS

(SELECT STATEMENT) ,

<cte-name2>

AS

(SELECT STATEMENT)

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

SELECT / INSERT / UPDATE / DELETE

Ex 1 :- delete first 3 rows from emp ?

WITH E

AS

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

EMPNO,ENAME,SAL

FROM EMP )

DELETE FROM E WHERE RNO <= 3

Ex 2 :- delete duplicate rows ?

EMP33

ENO ENAME SAL

1 A 5000

2 B 6000

3 C 7000

1 A 5000 => duplicate

2 B 6000 => duplicate

STEP 1 :- generate row numbers with in the group of eno,ename,sal

SELECT ENO,ENAME,SAL,

ROW\_NUMBER() OVER (PARTITION BY ENO,ENAME,SAL

ORDER BY ENO ASC) AS RNO

FROM EMP33

1 A 5000.00 1

1 A 5000.00 2

2 B 6000.00 1

2 B 6000.00 2

3 C 7000.00 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 EMP33)

DELETE FROM E WHERE RNO > 1

=>

T1 T2

F1 C1

1 A

2 B

3 C

output :-

1 A

2 B

3 C

=>

T1

AMT

1000

-200

3000

-800

2000

-500

output :-

POS NEG

1000 -200

3000 -800

2000 -500

scalar sub-queries :-

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

=> sub-queries in select clause are called scalar sub-queries

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

FROM tabname

WHERE cond

=> sub-query output acts like a column for outer query

=> use scalar sub-query to show the query output in seperate column

Ex 1 :-

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

(SELECT COUNT(\*) FROM DEPT) AS DEPT

EMP DEPT

11 4

Ex 2 :-

=> display dept wise total sal ?

SELECT DEPTNO,SUM(SAL) AS DEPT\_TOTSAL

FROM EMP

GROUP BY DEPTNO

10 8750.00

20 10075.00

30 6550.00

=> display DEPTNO DEPT\_TOTSAL TOTSAL ?

SELECT DEPTNO,SUM(SAL) AS DEPT\_TOTSAL ,

(SELECT SUM(SAL) FROM EMP) AS TOTSAL

FROM EMP

GROUP BY DEPTNO

10 8750.00 25375.00

20 10075.00 25375.00

30 6550.00 25375.00

=> display DEPTNO DEPT\_TOTSAL TOTSAL PCT ?

PCT = (DEPT\_TOTSAL/TOTSAL)\*100

SELECT DEPTNO,SUM(SAL) AS DEPT\_TOTSAL ,

(SELECT SUM(SAL) FROM EMP) AS TOTSAL ,

(SUM(SAL) / (SELECT SUM(SAL) FROM EMP))\*100 AS PCT

FROM EMP

GROUP BY DEPTNO

10 8750.00 25375.00 34.48

20 10075.00 25375.00 39.70

30 6550.00 25375.00 25.81

select stmt

where

order by

distinct

top

functions

group by

joins

set operators

sub-queries

integrity constraints

15-jul-24

DATABASE TRANSACTIONS :-

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

=> a transaction is a unit of work that contains one or more dmls (insert,update,delete)

and must be saved as a whole or must be cancelled as a whole.

ex :- money transfer

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

update1 update2

(bal = bal - 1000) (bal = bal + 1000)

successful failed INVALID

failed successful INVALID

successful successful VALID

failed failed VALID

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

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

1 COMMIT => to save transaction

2 ROLLBACK => to cancel transaction

3 SAVE TRANSACTION => to cancel part of the transaction

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

=> In SQL server a transaction begins implicitly with dml command and ends inplicitly with commit

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

ex 1 :-

CREATE TABLE a(a INT) => txn ends with commit

INSERT INTO a VALUES(10) => txn ends with commit

INSERT INTO a VALUES(20) => txn ends with commit

INSERT INTO a VALUES(30) => txn ends with commit

INSERT INTO a VALUES(40) => txn ends with commit

ROLLBACK => error => trying to end without starting

output :- all operations are implicitly committed

ex 2 :-

CREATE TABLE a(a INT) => implicitly committed

BEGIN TRANSACTION => txn begins T1

INSERT INTO a VALUES(10)

INSERT INTO a VALUES(20)

INSERT INTO a VALUES(30)

INSERT INTO a VALUES(40)

ROLLBACK => txn ends

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

ex 3 :-

CREATE TABLE a(a INT) => implicitly committed

BEGIN TRANSACTION => txn begins T1

INSERT INTO a VALUES(10)

INSERT INTO a VALUES(20)

COMMIT => txn ends

INSERT INTO a VALUES(30) => implicitly committed

INSERT INTO a VALUES(40) => implicitly committed

ROLLBACK => ERROR

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

SAVE TRANSACTION :-

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

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

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

ex :-

CREATE TABLE a(a INT)

BEGIN TRANSACTION

INSERT INTO a VALUES(10)

INSERT INTO a VALUES(20)

SAVE TRANSACTION ST1

INSERT INTO a VALUES(30)

INSERT INTO a VALUES(40)

SAVE TRANSACTION ST2

INSERT INTO a VALUES(50)

INSERT INTO a VALUES(60)

ROLLBACK TRANSACTION ST2

SELECT \* FROM a

10

20

30

40

16-jul-24

DATABASE SECURITY :-

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

1 logins => provides security at server level

2 users => provides security at db level

3 privileges => provides security at table level

4 views => provides security at row & col level

SERVER (LOGINS)

DATABASE (USERS)

TABLE (PRIVILEGES)

ROWS & COLS (VIEWS)

LOGIN

SA

NARESH

DB11AM

DBO (SA)

EMP

DEPT

STUDENT

CUST

VIJAY (NARESH)

CREATING LOGINS IN SQL SERVER :-

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

=> In object explorer select logins => new login

Enter login name :- NARESH

select SQL server authentication

Enter password :- 123

=> click OK

CREATEING USER IN DB :-

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

=> expand the db in which you want to create user

DB4PM

SECURITY

USER => NEW USER

=> Enter Username :- VIJAY

Enter Login :- NARESH

=> click OK

note :- tables created by DBO cannot be accessed by VIJAY , if VIJAY wants to

access tables created by DBO then VIJAY needs permissions.

PRIVILEGES :-

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

=> privileges means permissions

=> permissions are granted by owner using GRANT command

syn :- GRANT <privileges> ON <tabname> TO <usernames>

Ex :-

DBO :-

----------

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

VIJAY :-

-----------

1 SELECT \* FROM EMP

2 UPDATE EMP SET SAL =2000 WHERE EMPNO = 7369

3 DELETE FROM EMP WHERE EMPNO = 7844

NOTE :- changes made by VIJAY are visible to DBO

REVOKE command :-

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

=> command used to take back permissions from user

syn :- REVOKE <privileges> ON <TABNAME> FROM <USERNAME>

Ex :-

DBO :-

----------

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

DB objects :-

==========

SQL :-

=======

1 TABLES

2 VIEWS

3 SYNONYMS

4 SEQUENCES

5 INDEXES

T-SQL :-

=======

6 STORED PROCEDURES

7 FUNCTIONS

8 TRIGGERS

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 is a representation of a query

=> views are created

1 to provide security

2 to reduce complexity

=> views are 2 types

1 simple views

2 complex views

simple views :-

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

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

CREATE VIEW <NAME>

AS

SELECT STATEMENT

ex :-

CREATE VIEW V1

AS

SELECT EMPNO,ENAME,JOB,HIREDATE,DEPTNO FROM EMP

=> SQL server creates view "v1" and stores query but not query output

SELECT \* FROM V1

=> when above query submitted to SQL server , it executes the query as follows

SELECT \* FROM (SELECT EMPNO,ENAME,JOB,HIREDATE,DEPTNO FROM EMP)

providing security :-

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

=> with the help of views we can grant specific rows & cols to users .

DBO :-

----------

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

VIJAY :-

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

1 SELECT \* FROM V1

2 UPDATE V1 SET JOB='CLERK' WHERE EMPNO = 7566

3 UPDATE V1 SET SAL=4000 WHERE EMPNO = 7566 => ERROR

4 INSERT INTO V1 VALUES(999,'ABC','CLERK',GETDATE(),10)

ROW LELVEL SECURITY :-

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

CREATE VIEW V2

AS

SELECT EMPNO,ENAME,JOB,DEPTNO

FROM EMP

WHERE DEPTNO = 20

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

VIJAY :-

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

INSERT INTO V2 VALUES(888,'KLM','MANAGER',30)

above insert command executed successfully even though it is violating

where condition.

WITH CHECK OPTION :-

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

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

view violates where condition that dml is not accepted

CREATE VIEW V3

AS

SELECT EMPNO,ENAME,JOB,DEPTNO

FROM EMP

WHERE DEPTNO = 20

WITH CHECK OPTION

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

VIJAY :-

---------

INSERT INTO V3 VALUES(777,'PQR','CLERK',30) => ERROR

17-jul-24

Complex views :-

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

=> a view said to be complex

1 if it is based on multiple tables

2 it query contains group by clause

distinct clause

aggregate functions

set operators

sub-queries

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

ex 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 employee details with dept details

then instead of writing join query write the simple query

SELECT \* FROM CV1

ex 2 :-

CREATE VIEW CV2

AS

SELECT D.DNAME,MAX(E.SAL) AS MAXSAL,

MIN(E.SAL) AS MINSAL,

SUM(E.SAL) AS TOTSAL,

COUNT(\*) AS CNT

FROM EMP AS E INNER JOIN DEPT AS D

ON E.DEPTNO = D.DEPTNO

GROUP BY D.DNAME

=> aftrer creating view , whenever we want dept wise summary then execute the following query

SELECT \* FROM CV2

=> difference between simple and complex views ?

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

=> list of tables & views ?

SELECT \* FROM INFORMATION\_SCHEMA.TABLES

=> list of views ?

SELECT \* FROM INFORMATION\_SCHEMA.VIEWS

=> display only tables list ?

SELECT TABLE\_NAME,TABLE\_TYPE

FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_TYPE='BASE TABLE'

Droping :-

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

DROP VIEW V1

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

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

SQL> DROP TABLE products;

What is the implication of this command? (Choose all that apply.)

A. All data along with the table structure is deleted.

B. The pending transaction in the session is committed.

C. All indexes on the table will remain but they are invalidated.

D. All views and synonyms will remain but they are invalidated.

E. All data in the table are deleted but the table structure will remain.

synonyms :-

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

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

=> if tablename is lengthy then we can give a simple and short name to the table

called synonym , 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 = 7369

=>

1 CREATE SYNONYM E FOR EMP

2 SELECT \* FROM EMP AS E

3 SP\_RENAME 'EMP','E'

=> difference between synonym and alias ?

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

=> list of synonyms ?

SELECT NAME,BASE\_OBJECT\_NAME FROM SYS.SYNONYMS

Droping :-

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

DROP SYNONYM E

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

ans :- synonyms are not dropped but cannot be queried

SERVER

DATABASE

USER

TABLE

ROWS & COLS

CONSTRAINTS

INDEXES

TRIGGERS

VIEWS

SYNONYM

18-jul-24

SEQUENCES :-

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

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

=> used to auto increment column values.

CREATE SEQUENCE <NAME>

[START WITH <value>]

[INCREMENT BY <value>]

[MAXVALUE <value>]

[MINVALUE <value>]

[CYCLE]

ex 1 :-

CREATE SEQUENCE S1

START WITH 1

INCREMENT BY 1

MAXVALUE 5

CREATE TABLE STUDENT(SID INT,SNAME VARCHAR(10))

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1 , 'A')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1 , 'B')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1 , 'C')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1 , 'D')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1 , 'E')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1 , 'F') => ERROR

SELECT \* FROM STUDENT

SID SNAME

1 A

2 B

3 C

4 D

5 E

ex 2 :-

CREATE SEQUENCE S2

START WITH 100

INCREMENT BY 1

MAXVALUE 999

=> use above sequence to update empno ?

UPDATE EMP SET EMPNO = NEXT VALUE FOR S2

ex 3 :-

BILL

BILLNO BDATE AMT

DMART/0724/1

DMART/0724/2

CREATE SEQUENCE S3

START WITH 1

INCREMENT BY 1

MAXVALUE 9999

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

INSERT INTO BILL

VALUES('DMART/' +

FORMAT(GETDATE(),'MMyy') + '/' +

CAST(NEXT VALUE FOR S3 AS VARCHAR) ,GETDATE(),3000)

SELECT \* FROM BILL

DMART/0724/1 2024-07-18 11:57:46.663 1000.00

DMART/0724/2 2024-07-18 11:57:56.427 2000.00

DMART/0724/3 2024-07-18 11:58:01.050 3000.00

How to restart sequence :-

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

1 manually

2 cycle option

manually :-

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

ALTER SEQUENCE S1 RESTART WITH 1

using cycle option :-

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

=> by default sequence created with nocycle.

=> if created with NOCYCLE then it starts from start with and generates upto max and after

reachng max then it stops.

=> if created with CYCLE then it starts from start with and generates upto max and after

reaching max then it will reset to min.

CREATE SEQUENCE S4

START WITH 1

INCREMENT BY 1

MAXVALUE 5

MINVALUE 1

CYCLE

=> list of sequences ?

SELECT \* FROM INFORMATION\_SCHEMA.SEQUENCES

Droping sequences :-

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

DROP SEQUENCE S1

IDENTITY :-

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

=> IDENTITY is also used to generate sequence numbers

IDENTITY(SEED,INCR)

SEED => start

default 1

INCR => increment

default 1

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')

SELECT \* FROM CUST

CID CNAME

100 A

101 B

102` C

=> difference between identity and sequence ?

|  |  |  |
| --- | --- | --- |
|  | identity | sequence |
| 1 | identity tied to a particular column in particular table | not tied to a particular column |
| 2 | not declared with maximum | created with maximum |
| 3 | generates value whenever a new row is inserted | generate value by calling  next value for sequence |
| 4 | can be reset by using dbcc command | can be reset by using cycle option or alter command |

How to reset identity :-

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

DBCC CHECKIDENT(tablanem,reseed,value)

ex :-

DBCC CHECKIDENT('CUST',RESEED,99)

19-jul-24

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

particulat topic can be located fastly but In db using index a particular record

can be located fastly.

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

=> Indexes created on columns

1 frequently used in where clause

2 frequently used in join operation

Types of Indexes :-

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

1 Non Clustered

simple

composite

unique

2 Clustered

simple non clustered index :-

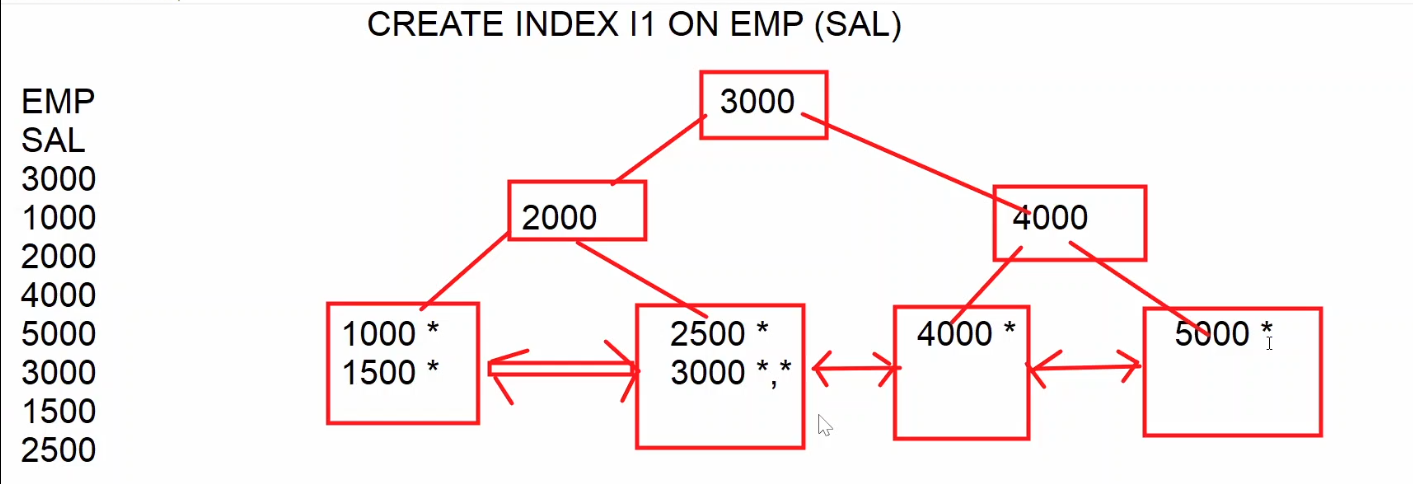
---------------------------------------

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

CREATE INDEX <NAME> ON <TABNAME>(COLNAME)

Ex :- Note \* indicates Address and stored in index, of 2 \* then two records with same salary and two indexes will be stored in index

CREATE INDEX I1 ON EMP (SAL)



SELECT \* FROM EMP WHERE SAL = 3000 (INDEX)

SELECT \* FROM EMP WHERE SAL >= 3000 (INDEX)

SELECT \* FROM EMP WHERE SAL <= 3000 (INDEX)

SELECT \* FROM EMP WHERE ENAME='blake' (TABLE)

SELECT \* FROM EMP (TABLE)

COMPOSITE INDEX :-

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

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

CREATE INDEX I2 ON EMP (DEPTNO,JOB)

20

10 30

10 CLERK \* 20 ANALYST \*,\* 30 CLERK \*

10 MGR \* 20 CLERK \*,\* 30 MGR \*

20 MGR \* 30 SALESMAN \*,\*,\*,\*

=> SQL SERVER uses above index when where cond based on leading column of the index

i.e. deptno.

SELECT \* FROM EMP WHERE DEPTNO = 20 (INDEX)

SELECT \* FROM EMP WHERE DEPTNO = 20 AND JOB = 'CLERK' (INDEX)

SELECT \* FROM EMP WHERE JOB='CLERK' (TABLE)

UNIQUE INDEX :-

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

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

CREATE UNIQUE INDEX I3 ON EMP(ENAME)

k

g q

adams \* james \* martin \* scott \*

allen \* jones \* miller \* smith \*

blake \*

1 SELECT \* FROM EMP WHERE ENAME='blake'

2 INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(100,'blake',4000) => ERROR

=> different methods to enforce uniqueness ?

1 primary key / unique constraint

2 unique index

=> primary key / unique columns are automatically indexed by SQL server and

SQL server creates a unique index on primary key / unique columns

and unique index doesn't allow duplicates , so primary key / unique

also doesn't allow duplicates.

CLUSTERED INDEX :-

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

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

actual records.

EX :-

CREATE TABLE CUST(CID INT,CNAME VARCHAR(10))

CREATE CLUSTERED INDEX I5 ON CUST(CID)

INSERT INTO CUST VALUES(10,'A')

INSERT INTO CUST VALUES(80,'B')

INSERT INTO CUST VALUES(40,'C')

INSERT INTO CUST VALUES(60,'D')

50

30 60

10 A 40 C 60 D 80 B

1 SELECT \* FROM CUST WHERE CID = 60

2 SELECT \* FROM CUST => SQL server goes to index and access

all leaf nodes from left to right

10 A

40 C

60 D

80 B

=> only one clustered index allowed per table

=> SQL server implicitly creates a clustered index on primary key column

CLUSTERED VS NON CLUSTERED indexes :-

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

|  |  |  |
| --- | --- | --- |
|  | **non clustered** | **clustered** |
| 1 | stores pointers to actual records | stores actual records |
| 2 | order of the records in table and order of the records in index are not same | order of the records in table and index are same |
| 3 | index and table are two separate objects | index and table are one object |
| 4 | requires extra storage | doesn't need extra storage |
| 5 | requires two lookups to get specific row | requires single lookup |
| 6 | SQL server allows 999 non clustered indexes per table | only one clustered index allowed per table |
| 7 | by default, non-clustered index created on unique column | by default, a clustered index created on primary key column |

=> list of indexes created on table ?

SP\_HELPINDEX EMP

============================================================================

CREATING NEW TABLE FROM EXISTING TABLE (replica) :-

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

SELECT columns / \* INTO <new-tabname>

FROM <old-tabname>

[WHERE condition]

ex 1 :- (copying all rows and all cols)

SELECT \* INTO EMP10

FROM EMP

=> a new table is created with rows & cols return by query

ex 2 :- copying specific rows & cols

SELECT EMPNO,ENAME,JOB,SAL INTO EMP11

FROM EMP

WHERE JOB IN ('CLERK','MANAGER')

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

SELECT \* INTO EMP12

FROM EMP

WHERE 1=2

ex 4 :- copy table from one db to another db

copy emp table from DB11AM to HR db ?

SELECT \* INTO HR.DBO.EMP

FROM DB11AM.DBO.EMP

copying data from one table to another table :-

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

INSERT INTO <TARGET-TABLE>

SELECT COLUMNS / \* FROM <SOURCE-TABLE>

Ex :-

=> copy data from emp to emp12 ?

INSERT INTO EMP12

SELECT \* FROM EMP

MERGE command :-

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

=> command used to merge data into a table.

=> merge is the combination of insert , update & delete.

=> merge is used to manage replicas.

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

syntax :-

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 ;

22-jul-24

example :-

step 1 :- create source table

create table custs

(

cid int ,

cname varchar(10),

city char(3)

)

insert into custs values(10,'A','HYD') , (11,'B','MUM')

step 2 :- create target table

select \* into custt from custs

step 3 :- modify source table

insert into custs values(12,'C','DEL')

update custs set city = 'BLR' where cid = 10

step 4 :- apply changes made to custs to custt ?

MERGE INTO CUSTT AS T

USING CUSTS AS S

ON (S.CID = T.CID)

WHEN MATCHED THEN

UPDATE SET T.CITY = S.CITY

WHEN NOT MATCHED THEN

INSERT VALUES(S.CID,S.CNAME,S.CITY)

WHEN NOT MATCHED BY SOURCE THEN

DELETE ;

ex 2 :-

EMPS EMPT

EMPID ENAME SAL EMPID ENAME SAL

100 A 5000 100 A

101 B 6000 101 B

102 C 7000 102 C

=> copy salaries from EMPS to EMPT ?

1 update command

2 merge command

===================================================================

PIVOT operator / CROSS-TAB Query :-

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

=> used to convert rows into columns

=> used to display data in matrix form.

syn :-

SELECT columns

FROM (SELECT required data FROM tabname) AS <ALIAS>

PIVOT

(

aggr-expr FOR COLNAME IN (V1,V2,V3----)

) AS <pivot-table-name>

ORDER BY colname ASC/DESC

Ex 1 :-

10 20 30

ANALYST ? 6000 ?

CLERK ? ? ?

MANAGER ? ? ?

SALESMAN ? ? 5600

SELECT \*

FROM ( SELECT DEPTNO,JOB,SAL FROM EMP) AS E

PIVOT

(

SUM(SAL) FOR DEPTNO IN ([10],[20],[30])

) AS PIVOT\_TBL

ORDER BY JOB ASC

Ex 2 :-

1 2 3 4

1980 ? ? ? ?

1981 ? ? ? ?

1982

1983

SELECT \*

FROM (SELECT DATEPART(YY,HIREDATE) AS YEAR,

DATEPART(QQ,HIREDATE) AS QRT,

EMPNO

FROM EMP) AS E

PIVOT

(

COUNT(EMPNO) FOR QRT IN ([1],[2],[3],[4])

) AS PIVOT\_TBL

ORDER BY YEAR ASC

Ex 3 :-

STUDENT

SNO SNAME SUBJECT MARKS

1 A MAT 80

1 A PHY 70

1 A CHE 60

2 B MAT 60

2 B PHY 70

2 B CHE 50

output :-

SNO SNAME MAT PHY CHE

1 A 80 70 60

2 B 60 70 50

SQL

COMMANDS CLAUSES OPERATIONS OBJECTS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DDL WHERE FILTERING TABLES

DML ORDER BY SORTING VIEWS

DQL DISTINCT ELIMINATING DUPLICATES SYNONYMS

TCL TOP TOP N ROWS SEQUENCES

DCL GROUP BY GROUPING INDEXES

ON JOINS

SUB-QUERIES

FUNCTIONS

=================================================================

T-SQL

======

1 basic t-SQL programming

2 conditional stmts

3 loops

4 cursors

5 error handling

6 stored procedures

7 functions

8 triggers

9 dynamic sql

SQL server

non-procedural procedural

SQL T-SQL

(commands) (blocks)

Features :-

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

1 improves performance :-

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

=> In t-SQL we can group SQL commands into one block and we submit that block to SQL server ,

so in t-SQL no of requests and response between user and SQL server are reduced and

performance is improved.

2 supports conditional statements :-

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

=> t-SQL supports conditional statements like if-else , so in t-SQL we can execute

SQL commands based on conditions.

3 supports loops :-

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

=> t-SQL supports looping statements like while , so with the help of loops we

can execute SQL commands repeatedly multiple times.

4 supports error handling :-

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

=> In t-SQL if any statement causes error then we can handle that error and we

can replace system generate message with our own simple and user

friendly message.

5 supports reusability :-

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

=> t-SQL programs can be stored in db , so applications which are connected to db

can reuse t-SQL programs.

=> T-SQL blocks are 2 types

1 anonymous blocks

2 named blocks

stored procedures

functions

triggers

Anonymous Blocks :-

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

=> a t-SQL program without name is called anonymous block

=> the following statements are used in t-SQL programs

1 DECLARE

2 SET

3 PRINT

DECLARE :-

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

=> statement used to declare variables

syn :- DECLARE @varname datatype(size)

ex :-

DECLARE @x INT

DECLARE @s VARCHAR(10)

DECLARE @d DATE

OR

DECLARE @x INT,@s VARCHAR(10),@d DATE

SET :-

----------

=> statement used to assign value to variable

SET @varname = value

ex :-

SET @x = 100

SET @s = 'welcome'

SET @d = getdate()

SET @x = 100 , @s = 'welcome ' , @d = getdate() => invalid

PRINT :-

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

=> statement used to print messages / values

PRINT message / variable

ex :-

PRINT 'hello'

PRINT @x

examples :-

=> write a prog to add two numbers ?

DECLARE @a INT,@b INT,@c INT

SET @a = 100

SET @b = 200

SET @c = @a + @b

PRINT @c

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

DECLARE @d DATE

SET @d = GETDATE()

PRINT DATENAME(DW,@d)

DB PROGRAMMING WITH T-SQL :-

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

=> from t-SQL to work with 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)

SELECT stmt syntax :-

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

SELECT @var1 = col1 ,

@var2 = col2 ,

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

FROM tabname

WHERE cond

ex :-

SELECT @n = ename,

@s = sal

FROM emp

WHERE empno = 105

24-jul-24

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

DECLARE @eno INT,@name VARCHAR(10),@sal MONEY

SET @eno = 7788

SELECT @name = ename ,@sal = sal FROM emp WHERE empno = @eno

PRINT @name + ' ' + CAST(@sal AS VARCHAR)

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

DECLARE @eno INT,@hire DATE,@expr TINYINT

SET @eno = 7844

SELECT @hire = hiredate FROM emp WHERE empno = @eno

SET @expr = DATEDIFF(YY,@hire,getdate())

PRINT ' EXPERIENCE = ' + CAST(@expr AS VARCHAR) + ' YEARS'

Conditional statements :-

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

1 IF-ELSE

2 MULTI IF

3 NESTED IF

IF-ELSE :-

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

IF COND

BEGIN

statements

END

ELSE

BEGIN

statements

END

MULTI IF :-

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

IF COND1

BEGIN

statements

END

ELSE IF COND2

BEGIN

statements

END

ELSE IF COND3

BEGIN

statements

END

ELSE

BEGIN

statements

END

NESTED IF :-

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

IF COND

BEGIN

IF COND

BEGIN

statements

END

ELSE

BEGIN

statements

END

END

ELSE

BEGIN

statements

END

examples :-

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

after increment if sal exceeds 5000 then cancel that increment ?

DECLARE @eno INT,@amt MONEY,@sal MONEY

SET @eno = 7788

SET @amt = 2500

BEGIN TRANSACTION

UPDATE emp SET sal = sal + @amt WHERE empno = @eno

SELECT @sal = sal FROM emp WHERE empno = @eno

IF @sal > 5000

ROLLBACK

ELSE

COMMIT

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

if job=clerk incr by 10%

salesman 15%

manager 20%

others 5%

DECLARE @eno INT ,@job VARCHAR(10),@pct TINYINT

SET @eno = 7844

SELECT @job=job FROM emp WHERE empno = @eno

IF @job='CLERK'

SET @pct = 10

ELSE IF @job='SALESMAN'

SET @pct = 15

ELSE IF @job='MANAGER'

SET @pct = 20

ELSE

SET @pct = 5

UPDATE emp SET sal = sal + (sal\*@pct/100) WHERE empno = @eno

SQL query :-

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

UPDATE EMP

SET SAL = CASE JOB

WHEN 'CLERK' THEN sal + (sal\*0.1)

WHEN 'SALESMAN' THEN sal + (sal\*0.15)

WHEN 'MANAGER' THEN sal + (sal\*0.2)

ELSE sal + (sal\*0.05)

END

=> write a prog to process bank transaction (W/D) ?

ACCOUNTS

ACCNO ACTYPE BAL

100 S 10000

101 S 20000

DECLARE @acno INT ,@type CHAR(1),@amt MONEY,@bal MONEY

SET @acno = 100

SET @type='W'

SET @amt = 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'

=> wap 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

25-jul-24

=> wap 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 TINYINT,@s2 TINYINT,@s3 TINYINT

DECLARE @total INT,@avg DECIMAL(5,2),@result CHAR(4)

SET @sno = 1

SELECT @s1 = s1 , @s2 = s2,@s3= s3 FROM student WHERE sno = @sno

SET @total = @s1 + @s2 + @s3

SET @avg = @total/3.0

IF @s1>=35 AND @s2 >=35 AND @s3>=35

SET @result='pass'

ELSE

SET @result='fail'

INSERT INTO result VALUES(@sno,@total,@avg,@result)

WHILE loop :-

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

=> loops are used to execute statements repeatedly multiple times

WHILE(cond)

BEGIN

statements

END

if cond = true loop continues

if cond = false loop terminates

examples :-

=> wap to print numbers from 1 to 20 ?

DECLARE @x TINYINT = 1

WHILE(@x <= 20)

BEGIN

PRINT @x

SET @x = @x + 1

END

=> wap to print 2024 calendar ?

DATE DAY

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

=> wap 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'

/\* to find first sunday \*/

WHILE(DATENAME(DW,@d1) <> 'SUNDAY')

BEGIN

SET @d1 = DATEADD(DD,1,@d1)

END

/\* to print sundays \*/

WHILE(@d1 <= @d2)

BEGIN

PRINT CAST(@d1 AS VARCHAR) + ' ' + DATENAME(DW,@d1)

SET @d1 = DATEADD(DD,7,@d1)

END

=> wap to print following pattern ?

input :- NARESH

output :-

N

A

R

E

S

H

DECLARE @s VARCHAR(10) = 'NARESH' , @x TINYINT = 1

WHILE(@x <= LEN(@s))

BEGIN

PRINT SUBSTRING(@s,@x,1)

SET @x = @x + 1

END

=> wap to print following pattern ?

input :- NARESH

output :-

N

NA

NAR

NARE

NARES

NARESH

DECLARE @s VARCHAR(10) = 'NARESH' , @x TINYINT = 1

WHILE(@x <= LEN(@s))

BEGIN

PRINT SUBSTRING(@s,1,@x)

SET @x = @x + 1

END

=> wap to input string and print reverse?

input :- NARESH

output :- HSERAN

DECLARE @s1 VARCHAR(10),@s2 VARCHAR(10)=''

DECLARE @x TINYINT

SET @s1 = 'NARESH'

SET @x = LEN(@s1)

WHILE(@x >= 1)

BEGIN

SET @s2 = @s2 + SUBSTRING(@s1,@x,1)

SET @x = @x-1

END

PRINT @s2

IF @s1 = @s2

PRINT 'palindrome'

ELSE

PRINT 'not a palindrome'

26-jul-24

CURSORS :-

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

=> cursors are used to process multiple records in t-SQL program.

=> using cursors we can access row-by-row in t-SQL program.

=> from t-SQL program if we submit a query to SQL server , it goes to db and

gets the data from db and copies that data into temporary memory

called cursor. In t-SQL program we can give name the cursor and

access row-by-row from cursor to the t-SQL program and process the row.

=> follow below steps to use cursor

1 DECLARE

2 OPEN

3 FETCH

4 CLOSE

5 DEALLOCATE

DECLARE :-

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

DECLARE <NAME> CURSOR FOR SELECT STATEMENT

Ex :-

DECLARE C1 CURSOR FOR SELECT ENAME,SAL FROM EMP

OPEN :-

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

OPEN <CURSOR-NAME>

Ex :-

OPEN C1

1 select stmt submitted to SQL server

2 SQL server executes the query and data returned by query is copied to cursor

3 c1 points to cursor

FETCH :-

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

=> "FETCH" statement is used to fetch record from cursor

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

EX :-

FETCH NEXT FROM C1 INTO vename,vsal ;

=> a fetch statement fetches one row at a time but to process multiple rows fetch stmt

should be executed multiple times , so fetch stmt should be in a loop.

CLOSE :-

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

CLOSE <cursor-name>

Ex :- CLOSE C1

DEALLOCATE :-

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

DEALLOCATE <cursor-name>

Ex :- DEALLOCATE C1

@@FETCH\_STATUS ;-

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

=> It is a system variable that returns fetch status

0 => if fetch successful

-1 => if fetch unsuccessful

=> fetch\_status is useful for loop condition

Examples :-

=> 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

=> wap to calculate total sal without using sum function ?

DECLARE C1 CURSOR FOR SELECT sal FROM emp

DECLARE @sal MONEY,@t MONEY = 0

OPEN C1

FETCH NEXT FROM C1 INTO @sal

WHILE(@@FETCH\_STATUS=0)

BEGIN

SET @t = @t + @sal

FETCH NEXT FROM C1 INTO @sal

END

PRINT @t

CLOSE C1

DEALLOCATE C1

=> wap 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

=> wap to find min sal ?

DECLARE C1 CURSOR FOR SELECT sal FROM emp

DECLARE @sal MONEY,@min MONEY

OPEN C1

FETCH NEXT FROM C1 INTO @min

WHILE(@@FETCH\_STATUS=0)

BEGIN

FETCH NEXT FROM C1 INTO @sal

IF @sal < @min

SET @min = @sal

END

PRINT @min

CLOSE C1

DEALLOCATE C1

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

=> wap 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,@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

SNO S1 S2 S3

1 80 90 70

2 30 60 50

=> wap to copy data from custs to custt ?

CUSTS

CID NAME

10 SACHIN RAMESH TENDULKAR

11 MAHENDRA SINGH DHONI

CUSTT

CID FNAME MNAME LNAME

=> wap to calculate total amount of particular order ?

ORDERS PRODUCTS

ORDID PRODID QTY PRODID PNAME PRICE

1000 100 2 100 A 1000

1000 101 1 101 B 500

1000 102 2 102 C 1500

1001 100 2

input ordid = 1000

output amount = 5500

27-jul-24

SCROLLABLE CURSOR :-

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

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

backward navigation.

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

backward navigation.

=> a forward only cursor supports only FETCH NEXT statement but a scrollable supports

the following fetch statements.

FETCH FIRST => fetches 1st row

FETCH NEXT => fetches next row

FETCH LAST => fetches last row

FETCH PRIOR => fetches previous row

FETCH ABSOLUTE N => fetches Nth row from 1st row

FETCH RELATIVE N => fetches Nth row from current row

Example1 :-

=> wap to print all employee 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

=> wap to print every 5th row ?

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

ERROR HANDLING / EXCEPTION HANDLING :-

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

1 syntax errors

2 logical errors

3 runtime errors (exceptions)

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

ex :- declare @x tinyint

set @x = 1000 => runtime error

print @x

=> if any statement causes runtime error then SQL server displays 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 then include a block called TRY----CATCH block

BEGIN TRY

statements => causes exception

END TRY

BEGIN CATCH

statements => handles exception

END CATCH

example :-

DECLARE @a TINYINT,@b TINYINT,@c TINYINT

BEGIN TRY

SET @a=100

SET @b=0

SET @c=@a/@b

PRINT @c

END TRY

BEGIN CATCH

PRINT 'ERROR----TRY AGAIN'

END CATCH

error handling functions :-

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

1 ERROR\_NUMBER() => returns error number

2 ERROR\_MESSAGE() => returns error message

Example 1 :-

DECLARE @a TINYINT,@b TINYINT,@c TINYINT

BEGIN TRY

SET @a=100

SET @b=0

SET @c=@a/@b

PRINT @c

END TRY

BEGIN CATCH

IF ERROR\_NUMBER() = 220

PRINT 'value exceeding limit'

ELSE IF ERROR\_NUMBER() = 8134

PRINT 'divisor cannot be zero'

ELSE

PRINT ERROR\_MESSAGE()

END CATCH

Example 2 :-

CREATE TABLE EMP66

(

EMPID INT PRIMARY KEY,

ENAME VARCHAR(10) NOT NULL,

SAL MONEY CHECK(SAL>=3000)

)

=> wap to insert data into emp66 table ?

DECLARE @eno INT,@name VARCHAR(10),@sal MONEY

BEGIN TRY

SET @eno=101

SET @name='B'

SET @sal = 1000

INSERT INTO EMP66 VALUES(@eno,@name,@sal)

END TRY

BEGIN CATCH

IF ERROR\_NUMBER() = 2627

PRINT 'empno should not be duplicate'

ELSE IF ERROR\_NUMBER() = 515

PRINT 'name should not be null'

ELSE IF ERROR\_NUMBER() = 547

PRINT 'sal >= 3000'

END CATCH

=> list of errors ?

select \* from sys.messages

USER DEFINED ERRORS :-

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

=> errors raised by user are called user defined errors

=> user raises exception to stop program execution based on some condition.

=> user raises exception by using RAISERROR built-in procedure

RAISERROR(error msg,severity level,state)

severity level => 0 to 25

0 to 10 => informational messages

11 to 18 => errors

19 to 25 => fatal errors

state => 0 to 255 => if same error raised at multiple locations using state

we can identity which part of the program causing

the error

Example :-

DECLARE @eno INT,@amt MONEY

SET @eno = 7844

SET @amt = 1000

IF DATENAME(DW,GETDATE())='SUNDAY'

RAISERROR('sunday not allowed',19,1)

ELSE

UPDATE EMP SET SAL = SAL + @amt WHERE EMPNO = @eno

==========================================================================

**29-jul-24**

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 procedures & functions we can divide a big t-sql program into small modules

2 reusability :-

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

=> procedures and functions can be stored in db , so applications which are connected to db

can reuse procedures & functions.

3 invoked from front-end :-

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

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

4 improves performance :-

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

=> proc/func improves performance because one time compilation 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 a 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.

CREATE OR ALTER PROCEDURE <name>

parameters if any

AS

statements

parameters :-

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

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

=> parameters can receives values coming from another program.

=> parameters are 2 types

1 INPUT (DEFAULT)

2 OUTPUT

=> INPUT always receives value from main program.

=> OUTPUT always sends value to main program.

main procedure

X -----------------------------------------> A (INPUT)

Y <----------------------------------------- B (OUTPUT)

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 :-

1 from ssms

2 from another t-sql prog

3 from front-end application

executing from ssms :-

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

EXECUTE procname

ex :-

EXECUTE RAISE\_SALARY

Example 2 :- procedures with INPUT 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

Execution :-

1 EXECUTE RAISE\_SALARY 7369,1000 positional association

2 EXECUTE RAISE\_SALARY @eno=7369,@amt=1000 named association

3 EXECUTE RAISE\_SALARY @amt=1000 , @eno = 7369

Example 3 :- procedure with output parameter

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

and after increment send the updated sal to calling program ?

CREATE OR ALTER PROCEDURE RAISE\_SALARY

@eno INT,

@amt MONEY,

@newsal MONEY OUTPUT

AS

UPDATE EMP SET SAL = SAL + @amt WHERE EMPNO = @eno

SELECT @newsal = SAL FROM EMP WHERE EMPNO = @eno

EXECUTION :-

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

DECLARE @S MONEY

EXECUTE RAISE\_SALARY 7369,1000,@S OUTPUT

PRINT @S

EXECUTE RAISE\_SALARY @eno=7369,@amt=1000,@newsal = @S OUTPUT

30-jul-24

Declaring parameters with default value :-

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

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

ex :- @amt money = 500

=> while calling procedure if we do not pass value to @amt then sql server assigns default value

Ex :-

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

=> create a procedure for money withdrawl ?

ACCOUNTS

ACCNO ACTYPE BAL

100 S 10000

101 S 20000

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

CREATE TABLE ACCOUNTS

(

ACCNO INT,

ACTYPE CHAR(1),

BAL MONEY)

INSERT INTO ACCOUNTS VALUES(100,'A',10000),(101,'B',20000)

CREATE TABLE TRANSACTIONS

(

TRID INT IDENTITY,

TTYPE CHAR(1),

TDATE DATETIME,

TAMT MONEY,

ACCNO INT

)

CREATE OR ALTER PROCEDURE DEBIT

@acno INT,

@amt MONEY,

@newbal MONEY OUTPUT

AS

DECLARE @bal MONEY

SELECT @bal = bal FROM accounts WHERE accno = @acno

IF @amt > @bal

RAISERROR('insufficient balance',16,1)

ELSE

BEGIN

BEGIN TRANSACTIONS

UPDATE accounts SET bal = bal - @amt WHERE accno = @acno

INSERT INTO transactions(ttype,tdate,tamt,accno)

VALUES('W',getdate(),@amt,@acno)

COMMIT

SELECT @newbal = bal FROM accounts WHERE accno = @acno

END

EXECUTION :-

DECLARE @B MONEY

EXECUTE DEBIT 100,1000,@B OUTPUT

PRINT @B

=> create procedure for money deposit ?

=> create procedure for money transfer ?

USER DEFINED FUNCTIONS :-

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

=> functions created by user are called user defined 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 block that accepts some input performs some

calculation and must return a value.

=> functions are 2 types

1 scalar valued functions (SVF)

2 table valued functions (TVF)

scalar valued functions :-

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

=> these functions returns one value

=> return type must be scalar types like int,varchar,date etc

=> return expression must be scalar variable

CREATE OR ALTER

FUNCTION <name>(parameters if any) RETURNS <type>

AS

BEGIN

statements

RETURN <expr>

END

Ex 1 :-

CREATE OR ALTER

FUNCTION CALC(@a INT,@b INT,@op CHAR(1)) RETURNS INT

AS

BEGIN

DECLARE @c INT

IF @op='+'

SET @c = @a + @b

ELSE IF @op='-'

SET @c = @a - @b

ELSE IF @op='\*'

SET @c = @a\*@b

ELSE

SET @c = @a/@b

RETURN @c

END

EXECUTION :-

1 sql commands

2 ssms

3 front-end

executing from sql commands :-

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

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

=> create a function to calculate experience of specific 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 SELECT DBO.GETEXPR(7844) => 43

2 SELECT EMPNO,ENAME,DBO.GETEXPR(EMPNO) AS EXPR FROM EMP

=> create a function to return total amount of particular order ?

ORDERS PRODUCTS

ordid prodid qty prodid pname price

1000 100 1 100 A 1000

1000 101 2 101 B 2000

1000 102 3 102 C 500

1001 100 2

INPUT ordid = 1000

OUTPUT amount = 6500

=> create a function that accepts deptno and returns name of the employees working for that dept ?

INPUT deptno = 20

OUTPUT SMITH,JONES,SCOTT,ADAMS,FORD

STRING\_AGG() :-

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

=> functions used to concatenate column values

STRING\_AGG(colname , seperator)

Ex :-

SELECT STRING\_AGG(ENAME,',') FROM EMP

31-jul-24

TABLE VALUED FUNCTIONS :-

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

=> these functions returns rows / records

=> return type of these functions must be TABLE.

=> return expression must be a select stmt

CREATE OR ALTER

FUNCTION <name> (parameters) RETURNS TABLE

AS

RETURN (select statement)

=> table valued functions allows only one stmt and it must be return statement.

example 1 :-

=> create a function that accepts deptno and returns list of employees working for the dept ?

CREATE OR ALTER

FUNCTION getEmpList(@dno INT) RETURNS TABLE

AS

RETURN (SELECT \* FROM EMP WHERE DEPTNO = @dno)

Execution :-

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

=> table valued functions are invoked in FROM clause.

SELECT \* FROM DBO.getEmpList(20)

Ex 2 :-

=> create a function that returns top n employees list based on sal ?

CREATE OR ALTER FUNCTION getTopNEmpList(@n INT) RETURNS TABLE

AS

RETURN (SELECT \*

FROM (SELECT EMPNO,ENAME,SAL,

DENSE\_RANK() OVER (ORDER BY SAL DESC) AS RNK

FROM EMP) AS E

WHERE RNK <= @n )

Execution :-

select \* from dbo.getTopNEmpList(5)

Assignment :-

ACCOUNTS

ACCNO ACTYPE BAL

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

create the following procedures & functions 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)

=> difference between procedures & functions ?

procedures functions

1 may or may not returns a value must return a value

2 can return multiple values always returns one value

3 returns values using output parameter returns value using return stmt

4 can contain dmls statements cannot contain dml statements

5 cannot be executed from can be executed from sql commands

sql commands

6 created to perform one or more created for calculations or to fetch data

dml operations on tables from table

=> difference between scalar and table valued functions ?

scalar table

1 returns one value returns records

2 return type must be scalar types return type must be always table

3 return expr is a scalar variable return expr is a select stmt

4 invoked in select clause invoked in from clause

=> list of procedures & functions created by user ?

SELECT \* FROM INFORMATION\_SCHEMA.ROUTINES

=> display DEBIT procedure code ?

SELECT ROUTINE\_DEFINITION

FROM INFORMATION\_SCHEMA.ROUTINES

WHERE ROUTINE\_NAME='DEBIT'

Droping :-

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

DROP PROCEDURE DEBIT

DROP FUNCTION CALC

31-jul-24

TRIGGERS :-

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

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

whenever user submits DML commands to sql server.

=> triggers are created

1 to control dmls

2 to enforce complex rules and validations

3 to audit day-to-day operations on tables

syntax :-

CREATE OR ALTER TRIGGER <NAME>

ON <TABNAME>

AFTER / INSTEAD OF INSERT,UPDATE,DELETE

AS

STATEMENTS

AFTER triggers :-

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

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

INSTEAD OF triggers :-

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

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

examples :-

=> 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 COMM = 800 WHERE EMPNO = 7839

=> 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

BEGIN

ROLLBACK

RAISERROR ('sunday not allowed',16,1)

END

=> create trigger to not to allow to update empno ?

CREATE OR ALTER TRIGGER T3

ON EMP

AFTER UPDATE

AS

IF UPDATE(EMPNO) OR UPDATE(HIREDATE)

BEGIN

ROLLBACK

RAISERROR(' cannot update',16,1)

END

Testing :-

UPDATE EMP SET HIREDATE = '2024-08-01' WHERE EMPNO = 7839 => error

Magic tables ;-

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

1 INSERTED

2 DELETED

=> these tables are called magic tables because they are created and deleted automatically

=> with the help of magic tables we can access data affected by dmls in triggers

=> 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,'ABC','CLERK',4000,------,20) => INSERTED

DELETE FROM EMP WHERE EMPNO = 7844 => DELETED

UPDATE EMP SET SAL = 2000 WHERE EMPNO = 7499 => INSERTED

EMPNO SAL

7499 2000

DELETED

EMPNO SAL

7499 1600

=> 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',16,1)

END

Testing :-

UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369

1 row is copied to inserted,deleted tables

2 executes update command

3 executes trigger

02-jul-24

=> create trigger to insert details into emp\_resign 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

INSERT INTO EMP\_RESIGN

SELECT EMPNO,ENAME,JOB,SAL,HIREDATE,GETDATE() FROM DELETED

Testing :-

DELETE FROM EMP WHERE EMPNO = 7369

1 row is copied to deleted table

2 executes delete command

3 executes trigger

Auditing :-

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

=> Auditing means monitoring day-to-day activities on tables

=> triggers are created to perform auditing

EMP\_AUDIT

UNAME OPERATION OPTIME NEW\_ENO NEW\_ENAME NEW\_SAL OLD\_ENO OLD\_ENAME OLD\_SAL

CREATE TABLE EMP\_AUDIT

(

UNAME VARCHAR(20),

OPERATION VARCHAR(10),

OPTIME DATETIME,

NEW\_ENO INT,

NEW\_ENAME VARCHAR(10),

NEW\_SAL MONEY,

OLD\_ENO INT,

OLD\_ENAME VARCHAR(10),

OLD\_SAL MONEY

)

CREATE OR ALTER TRIGGER T6

ON EMP

AFTER INSERT,UPDATE,DELETE

AS

DECLARE @OLDENO INT,@OLDENAME VARCHAR(10),@OLDSAL MONEY

DECLARE @NEWENO INT,@NEWENAME VARCHAR(10),@NEWSAL MONEY

DECLARE @OP VARCHAR(10)

DECLARE @CNT1 INT,@CNT2 INT

SELECT @CNT1 = COUNT(\*) FROM INSERTED

SELECT @CNT2 = COUNT(\*) FROM DELETED

IF @CNT1=1 AND @CNT2=0

SET @OP='INSERT'

ELSE IF @CNT1=0 AND @CNT2=1

SET @OP='DELETE'

ELSE

SET @OP = 'UPDATE'

SELECT @NEWENO=EMPNO,@NEWENAME=ENAME,@NEWSAL=SAL

FROM INSERTED

SELECT @OLDENO=EMPNO,@OLDENAME=ENAME,@OLDSAL=SAL

FROM DELETED

INSERT INTO EMP\_AUDIT

VALUES(USER\_NAME(),@OP,GETDATE(),

@NEWENO,@NEWENAME,@NEWSAL,

@OLDENO,@OLDENAME,@OLDSAL)

Testing :-

1 INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(100,'A',5000)

2 UPDATE EMP SET SAL=6000 WHERE EMPNO = 100

3 DELETE FROM EMP WHERE EMPNO = 100

4 SELECT \*FROM EMP\_AUDIT

11-DEC-23

INSTEAD OF trigger :-

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

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

Ex :-

=> create trigger to not to allow more than 4 employees per dept ?

EMP33

ENO ENAME DNO

1 A 10

2 B 10

3 C 10

4 D 10

5 E 10 => not allowed

CREATE OR ALTER TRIGGER T7

ON EMP33

INSTEAD OF INSERT

AS

DECLARE @ENO INT,@NAME VARCHAR(10),@DNO INT,@CNT INT

SELECT @ENO=ENO,@NAME=ENAME,@DNO = DNO FROM INSERTED

SELECT @CNT = COUNT(\*) FROM EMP33 WHERE DNO = @DNO

IF @CNT=4

RAISERROR('max 4 emps per dept',15,1)

ELSE

INSERT INTO EMP33 VALUES(@ENO,@NAME,@DNO)

Testing :-

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

INSERT INTO EMP33 VALUES(1,'A',10)

INSERT INTO EMP33 VALUES(2,'B',10)

INSERT INTO EMP33 VALUES(3,'C',10)

INSERT INTO EMP33 VALUES(4,'D',10)

INSERT INTO EMP33 VALUES(5,'E',10) => ERROR

AFTER INSTEAD OF

IF COND IF COND

BEGIN RAISE

ROLLBACK ELSE

RAISE DML

END

=> list of triggers created on tables ?

select trg.name,tab.name

from sys.triggers as trg INNER JOIN sys.tables as tab

on trg.parent\_id = tab.object\_id

T1 EMP

Droping triggers :-

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

drop trigger t1

03-aug-24

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 you want to execute should be passed as a string to

exec procedure.

=> dynamic sql is useful when we don't know table names and column names until

runtime.

example 1 :-

=> create a procedure to drop table ?

CREATE OR ALTER PROCEDURE DROP\_TABLE

@tname VARCHAR(100)

AS

EXEC (' DROP TABLE ' + @tname)

Testing :-

EXECUTE DROP\_TABLE 'EMP\_RESIGN'

=> create a 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(100)

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

Testing :-

EXECUTE DROP\_ALL\_TABLES

using sp\_executesql :-

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

sp\_executesql ' dynamic sql '

=> program to drop all views ?

declare c1 cursor for select table\_name from INFORMATION\_SCHEMA.VIEWS

declare @vname varchar(20) ,@str nvarchar(100)

open c1

fetch next from c1 into @vname

while(@@fetch\_status=0)

begin

set @str = 'DROP VIEW ' + @vname

execute sp\_executesql @str

fetch next from c1 into @vname

end

close c1

deallocate c1

=> wap to display no of rows in all tables ?

tablename no of rows

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

EMP 14

DEPT 4

declare c1 cursor for select table\_name from INFORMATION\_SCHEMA.TABLES

declare @tname varchar(20) ,@str nvarchar(100),@CNT INT

open c1

fetch next from c1 into @tname

while(@@fetch\_status=0)

begin

set @str = 'SELECT @X = COUNT(\*) FROM ' + @tname

execute sp\_executesql @str ,N'@X INT OUTPUT' , @X=@CNT OUTPUT

print @tname + ' ' + CAST(@CNT AS VARCHAR)

fetch next from c1 into @tname

end

close c1

deallocate c1

===========================================================================

5-aug-24

BACKUP & RESTORE :-

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

=> DB must be protected from different failures like hardware & software

=> with the help of backup we can protect db from failures

=> backup is the procedure that copies data from db to file (.bak file)

=> backup is useful to restore db when any failure occurs.

=> recovering db from backup is called restore

creating backup :-

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

=> select the db that you want to take backup

DB11AM => TASK => BACKUP

backup type :- FULL / DIFFERENTIAL

FULL => full db is copied to backup.

DIFFERENTIAL => since last backup whatever changes made to db only those

changes are copied to backup.

select destination => disk / url

add filename :- D:\BACKUP\db11am.bak

=> click OK

Command to take backup :-

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

=> open master database and execute the following command

backup database db11am to disk = 'D:\BACKUP\DB11AM.BAK'

How to take backup of all databases :-

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

CREATE OR ALTER PROCEDURE BACKUPDBS

AS

DECLARE C1 CURSOR FOR SELECT NAME FROM SYS.DATABASES WHERE DATABASE\_ID > 4

DECLARE @DBNAME VARCHAR(30) ,@FNAME VARCHAR(100)

OPEN C1

FETCH NEXT FROM C1 INTO @DBNAME

WHILE(@@FETCH\_STATUS=0)

BEGIN

SET @FNAME = 'D:\BACKUP\' + @DBNAME + '.BAK'

BACKUP DATABASE @DBNAME TO DISK = @FNAME

FETCH NEXT FROM C1 INTO @DBNAME

END

CLOSE C1

DEALLOCATE C1

EXECUTION :-

EXECUTE BACKUPDBS

Restoring Database :-

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

=> recovering database from backup is called restore

=> to restore database select Databases => Restore Database

=> select device

=> select backup filename

=> click ok

command to restore database :-

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

RESTORE DATABASE <name> FROM DISK = FILENAME

EX :-

RESTORE DATABASE DB11AM FROM DISK = 'D:\BACKUP\DB11AM.BAK'

=> what are temporary & global temporary tables ?

=> temporary tables are created to store data temporarly

=> used to store intermediate result

=> temporary tables are created in tempdb

=> temporary tables are 2 types

1 local temporary tables

2 global temporary tables

local temporary tables :-

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

=> created in one connection (session) and can be accessed only with in that connection

and cannot be accessed in another connection

=> local temporary tablenames are prefixed with "#"

method 1 :-

creating temporary table :-

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

CREATE TABLE #TEMP

(

EMPNO INT,ENAME VARCHAR(10),SAL MONEY,RNK INT

)

inserting data into temporary table :-

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

INSERT INTO #TEMP

SELECT EMPNO,ENAME,SAL,

DENSE\_RANK() OVER (ORDER BY SAL DESC) AS RNK

FROM EMP

display top 5 highest paid employees ?

SELECT \* FROM #TEMP WHERE RNK <= 5

method 2 :-

SELECT EMPNO,ENAME,SAL,

DENSE\_RANK() OVER (ORDER BY SAL DESC) AS RNK INTO #TEMP

FROM EMP

Global temporary tables :-

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

=> these tables are created in one­­ connection and can be accessed in another connection

=> global temporary tablenames are prefixed with ##

SELECT EMPNO,ENAME,SAL,

DENSE\_RANK() OVER (ORDER BY SAL DESC) AS RNK INTO ##TEMP

FROM EMP