SQL SERVER

Database :-

-----------

=> a Database is a organized collection of interrelated data.

for example a univ db stores data related to students,

courses,faculty et c.

Types of Databases :-

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

1 OLTP (online transaction processing db) / Operational DB

2 OLAP (online analytical processing db) / DWH / DSS

DWH => Data Warehouse

DSS => Decision Support System

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

and OLAP for data analysis.

=> OLTP is used for running business and OLAP is used for analyzing

business.

DBMS :- (Database Management System)

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

Evolution of DBMS :-

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

1960 FMS (File Mgmt system)

1970 HDBMS (Hierarchical DBMS)

NDBMS (Network DBMS)

1980 RDBMS (Relational DBMS)

1990 ORDBMS (Object Relational DBMS)

OODBMS (Object Oriented DBMS)

RDBMS :- (Relational Database Management System)

--------

=> rdbms concepts are introduced by E.F.CODD.

=> according to E.F.CODD in rdbms data must be organized in tables

i.e. rows and columns.

CUSTOMERS

CID NAME ADDR => columns,fields,atrributes

1 sachin mum

2 vijay hyd

3 rohit del => row,record,tuples

DB = collection of tables

TABLE = collection of rows and columns

ROW = collection of field values

COLUMN = collection of values assigned to one field

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

EX :- ACCNO,EMPID,AADHARNO,PANNO

=> one table is related to another table by using foreign key.

To establish relationship take pk of one table and that pk

to another table as fk.

ORDERS CUSTOMERS

ORDID ORD\_DT DEL\_DT CID(FK) CID(PK) NAME CADDR

1000 25- 30- 1 1 sachin mum

1001 25- 31- 2 2 vijay hyd

1002 26- 01- 3 3 rohit del

RDBMS softwares :- (SQL databases)

SQL SERVER

ORACLE

DB2

MYSQL

POSTGRESQL

NoSQL databases :-

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

mongoDB

cassandra

ORDBMS :- (object relational database management system)

=> it is the combination of RDBMS & OOPS

ORDBMS = RDBMS + OOPS (reusability)

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

=> ORDBMS supports reusability by using ADT (abstract datatype).

Example :-

SQL SERVER

ORACLE

summary :-

1 database

2 types of dbs

3 dbms

4 rdbms

5 ordbms

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

30-mar-21

SQL SERVER

=> SQL SERVER is basically RDBMS product from Microsoft and also supports

features of ORDBMS and used to create and to manage database.

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

Development Administration

creating tables installation of sql server

creating views creating database

creating syonyms creating logins

creating indexes db backup & restore

creating sequences db export & import

creating procedures db upgradation & migration

creating functions db mirroring & replication

creating triggers performance tuning

writing queries

Versions of SQL SERVER :-

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

version year

SQL SERVER 1.1 1991

SQL SERVER 4.2 1993

SQL SERVER 6.0 1995

SQL SERVER 6.5 1996

SQL SERVER 7.0 1998

SQL SERVER 2000 2000

SQL SERVER 2005 2005

SQL SERVER 2008 2008

SQL SERVER 2012 2012

SQL SERVER 2014 2014

SQL SERVER 2016 2016

SQL SERVER 2017 2017

SQL SERVER 2019 2019

`

sql server 2016 :-

1 polybase

2 json

3 temporal table to save data changes.

4 dynamic data masking and row level security

sql server 2017 :-

1 identity cache

2 New String functions

3 Automatic Tuning

sql server 2019 :-

1 Read, write, and process big data from Transact-SQL

2 Easily combine and analyze high-value relational data with high-volume big data.

3 Query external data sources.

4 Store big data in HDFS managed by SQL Server.

5 Query data from multiple external data sources through the cluster.

=> SQL SERVER is available in

1 on premises

2 on cloud

=> in " on premises " SQL SERVER db is deployed in the server

managed by client.

=> in " on cloud " SQL SERVER db is deployed in the server managed

by cloud service provider like amazon,microsoft etc.

=> the following components needs to be installed to work with

sql server.

1 SERVER

2 CLIENT

SERVER :-

---------

=> server is a system where sql server software is installed

and running. Inside the server using sql server software we can

create and manage database.

CLIENT :-

---------

=> client is also a system where users can

1 connects to server

2 submit requests to server

3 recieves response from server

client tool :-

SSMS (SQL SERVER MANAGEMENT STUDIO)

Download & Installation :-

sql server :-

1 download sql server 2017/2019

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

2 step by step installation of sql server

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

ssms :-

1 download ssms

https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

2 install ssms

https://www.guru99.com/sql-server-management-studio.html

06-apr-21

how to connect to sql server :-

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

=> to connect to sql server open ssms(sql server management studio)

and enter following details.

server type :- Database Engine

server name :- NARESHIT

authentication :- sql server authentication

login :- sa (system admin)

password :- 123

=> click connect

creating database in sql server :-

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

=> In object Explorer select Databases => New Database

Enter Database Name :- DB8AM

=> click OK

=> a New Database is created with following two files.

1 DATA FILE (.MDF)

2 LOG FILE (.LDF)

=> a Data File stores data and Log file stores operations.

Name Type Initital Size Auto Growth path

DB8AM DATA 8MB 64 MB C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\

DB8AM\_LOG LOG 8MB 64 MB C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\

=> Click OK

07-apr-21

TSQL (Transact-SQL)

=> SQL stands for structured query language

=> SQL is the language used to communicate with sql server.

=> user communicates with SQL SERVER by sending sql commands

called queries.

=> a query is command/instruction given to sql server to perform

some operation on database.

=> SQL is common to all databases

SQL SERVER ORACLE MYSQL DB2

SQL SQL SQL SQL

=> based on opertions over db , SQL is categorized into following

sublanguages.

DDL (Data Definition Language)

DML (Data Manipulation Language)

DQL (Data Query Language)

TCL (Transaction Control Language)

DCL (Data Control Language)

SQL

DDL DML DQL TCL DCL

create insert select commit grant

alter update rollback revoke

drop delete save transaction

truncate merge

empid ename sal DATA DEFINITION / METADATA

1 A 5000 DATA

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

client tool lang rdbms sofware storage

08-apr-21

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 tinyint decimal smallmoney date binary

smallint money time varbinary

char nchar int datetime varbinary(max)

varchar nvarchar bigint

varchar(max) nvarchar(max)

CHAR(size) :-

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

=> allows character data upto 8000 chars

=> recommended for fixed length char columns

EX :- NAME CHAR(10)

sachin----

wasted

rohit-----

wasted

=> in CHAR datatype extra bytes are wasted so don't use CHAR datatype

for variable length fields use it for fixed length fields.

EX :- gender char(1)

M

F

state\_code char(2)

AP

TS

MH

country\_code char(3)

IND

USA

AUS

VARCHAR(size) :-

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

=> allows character data upto 8000 chars

=> recommended for variable length fields.

=> in varchar datatype extra bytes are released

ex :- NAME VARCHAR(10)

sachin----

released

rohit-----

released

=> CHAR/VARCHAR allows ascii characters (256 chars) that includes

a-z,A-Z,0-9,special chars i.e. CHAR/VARCHAR allows alphanumeric

data.

ex :- VEHNO VARCHAR(10)

PANNO CHAR(10)

IFSC CHAR(10)

VARCHAR(MAX) :-

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

=> allows character data upto 2GB.

NCHAR/NVARCHAR/NVARCHAR(MAX) :-

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

=> allows unicode characters (65536 chars) that includes all

ascii chars and characters belongs to different languages.

Integer Types :-

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

=> allows exact numbers i.e. numbers without decimal part.

TINYINT 1 BYTE 0 TO 255

SMALLINT 2 BYTES -32768 TO 32767

INT 4 BYTES -2^31 TO 2^31-1

BIGINT 8 BYTES -2^63 TO 2^63-1

EX :- AGE TINYINT

EMPID SMALLINT

ACCNO INT

AADHARNO BIGINT

DECIMAL(P,S) :-

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

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

P => precision => total no of digits allowed

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

EX :- SALARY DECIMAL(7,2)

5000 => valid

5000.50 => valid

50000.50 => valid

500000.50 => invalid

09-apr-21 :-

currency types :-

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

=> currency types are used for fields related to money

smallmoney 4 bytes -214,748.3648 to 214,748.3647

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

ex :- fee smallmoney

sal money

bal money

DATE & TIME :-

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

DATE => allows only dates

TIME => allows only time

DATETIME => allows date & time

=> default date format in SQL SERVER is YYYY-MM-DD

=> default time format is HH:MI:SS

EX :- DOB DATE

LOGIN TIME

TXN\_DATE DATETIME

Binary Types :-

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

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

1 binary => allows binary data upto 8000 bytes

recommended for fixed length fields

extra bytes are wasted

2 varbinary => allows binary data upto 8000 bytes

recommended for variable length fields

extra bytes are released

3 varbinary(max) => allows binary data upto 2GB.

extra bytes are released

photo varbinary(max)

sign varbinary(5000)

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

CREATING TABLES IN SQL SERVER DB :-

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

syntax :-

CREATE TABLE <tabname>

(

col1 datatype(size),

col2 datatype(size),

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

)

Rules :-

---------

1 tabname should start with alphabet

2 tabname should not contain spaces and special chars but allows \_,#

3 tabname can be upto 128 chars

4 table can have 1024 columns

5 no of rows unlimited

tabname

123emp invalid

emp 123 invalid

emp\*123 invalid

emp123 valid

emp\_123 valid

Example :-

=> create table EMP with following structure ?

EMP

Empid Ename Job Sal Hiredate

CREATE TABLE emp

(

Empid TINYINT,

Ename VARCHAR(10),

Job VARCHAR(10),

Sal SMALLMONEY,

Hiredate DATE

)

SERVER

DB8AM

EMP

SP\_HELP :- command to see the structure of the table.

SP\_HELP <tabname>

SP\_HELP emp

column name datatype size

Empid tinyint 1

Ename varchar 10

Job varchar 10

Sal smallmoney 4

Hiredate date 3

10-apr-21

INSERTING DATA INTO TABLE :-

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

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

=> using INSERT command we can insert

1 single row

2 multiple rows

inserting single row :-

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

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

Ex :-

1 INSERT INTO emp VALUES(100,'sachin','clerk',4000,'2020-04-10')

2 INSERT INTO emp VALUES(101,'vijay','manager',8000,GETDATE())

inserting multiple rows :-

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

INSERT INTO emp

VALUES(102,'ravi','analyst',9000,GETDATE()),

(103,'sindhu','analyst',8000,GETDATE())

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 :- (explicit)

INSERT INTO emp VALUES(104,'satish',NULL,NULL,'2020-10-05')

method 2 :- (implicit)

INSERT INTO emp(empid,ename,hiredate) values(105,'vinod','2019-05-10')

INSERT INTO emp(empid,ename,hiredate)

VALUES(106,'kumar',GETDATE()),

(107,'ajay',GETDATE())

=> remaining two fields job,sal filled with NULLs.

Displaying Data :-

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

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

=> basic operation using SELECT command includes

1 PROJECTION => selecting specific columns

2 SELECTION => selecting specific rows

3 JOIN => combining data stored in multiple tables

syntax :- SELECT columns/\* FROM tabname

FROM clause => specifiy tablename

SELECT clause => specify column names

\* => all columns

SQL = ENGLISH

queries = sentences

clauses = words

=> display employee names and salaries ?

SELECT ename,sal FROM emp

=> display all the data from emp ?

SELECT \* FROM emp

14-apr-21

Operators in SQL SERVER :-

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

Arithmetic Operators => + - \* / %

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

Logical Operators => AND OR NOT

Special Operators => BETWEEN

IN

LIKE

IS

ANY

ALL

EXISTS

PIVOT & UNPIVOT

Set Operators => UNION

UNION ALL

INTERSECT

EXCEPT

WHERE clause :-

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

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

SELECT columns

FROM tabname

WHERE condition

condition :-

syn :- COLNAME OP VALUE

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

=> if cond = true then row is selected

=> if cond = false then row is not selected

=> display employee details whose empid=103 ?

SELECT \*

FROM emp

WHERE empid=103

=> display employee details whose name = vijay ?

SELECT \*

FROM emp

WHERE ename='vijay'

=> display employee details earning more than 5000 ?

SELECT \*

FROM emp

WHERE sal > 5000

=> display employee details joined after 2020 ?

SELECT \*

FROM emp

WHERE hiredate > 2020 => ERROR

SELECT \*

FROM emp

WHERE hiredate > '2020-12-31'

=> display employee details joined before 2020 ?

SELECT \*

FROM emp

WHERE hiredate < '2020-01-01'

15-APR-21

compound condition :-

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

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

compound condition.

WHERE cond1 AND cond2 Result

T T T

T F F

F T F

F F F

WHERE cond1 OR cond2 Result

T T T

T F T

F T T

F F F

=> display employees working as clerk,manager ?

SELECT \*

FROM emp

WHERE job='clerk','manager' => ERROR

SELECT \*

FROM emp

WHERE job='clerk' AND job='manager' => no rows

SELECT \*

FROM emp

WHERE job='clerk' OR job='manager'

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

SELECT \*

FROM emp

WHERE empid=100 OR empid=103 OR empid=105

IN operator :-

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

=> use IN operator when "=" comparision with multiple values or

list of values.

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

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

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

=> display employees working as clerk,manager ?

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

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

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

=> display employees not working as clerk,manager ?

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

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

SELECT \*

FROM emp

WHERE sal>5000 AND sal<10000

BETWEEN operator :-

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

=> use BETWEEN operator for range comparision

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

WHERE COLNAME NOT BETWEEN V1 AND V2

=> display employees earning between 5000 and 10000 ?

SELECT \* FROM emp WHERE sal BETWEEN 5000 AND 10000

=> display employees joined in 2020 year ?

SELECT \*

FROM emp

WHERE hiredate BETWEEN '2020-01-01' AND '2020-12-31'

=> display employees not joined in 2020 ?

SELECT \*

FROM emp

WHERE hiredate NOT BETWEEN '2020-01-01' AND '2020-12-31'

=> display employees working as clerk,manager and earning between

5000 and 10000 and joined in 2021 year ?

SELECT \*

FROM emp

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

AND

sal BETWEEN 5000 AND 10000

AND

hiredate BETWEEN '2021-01-01' AND '2021-12-31'

scenario :-

STUDENT

SNO SNAME S1 S2 S3

1 A 80 90 70

2 B 30 60 50

=> display list of students who are passed ?

SELECT \*

FROM student

WHERE s1>=35 AND s2>=35 AND s3>=35

=> display list of students who are failed ?

SELECT \*

FROM student

WHERE s1<35 OR s2<35 OR s3<35

=> display list of students who got distinction ?

SELECT \*

FROM student

WHERE (s1>=35 AND s2>=35 AND s3>=35)

AND

(s1+s2+s3)/3 >= 70

=> display list of students who got 1st class ?

=> display list of students who got 2nd class ?

=> display list of students who got 3rd class

=> display employees working as clerk,manager

and earning more than 2000 ?

SELECT \* FROM emp WHERE job='CLERK'

OR

job='MANAGER'

AND

sal>2000

=> above query returns clerk records earning less than 2000 because

operator AND has got more precedence than operator OR , to control

this use ( ).

SELECT \* FROM emp WHERE (

job='CLERK'

OR

job='MANAGER'

)

AND

sal>2000

LIKE operator :-

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

=> use LIKE opertor for pattern comparision

WHERE colname LIKE 'pattern'

WHERE colname NOT LIKE 'pattern'

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

wildcard characters :-

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

% => zero or many chars

\_ => exactly 1 char

=> display employee list name starts with 's' ?

SELECT \*

FROM emp

WHERE ename LIKE 's%'

=> display employee name ends with 's' ?

SELECT \*

FROM emp

WHERE ename LIKE '%s'

=> display employees name contains 's' ?

SELECT \*

FROM emp

WHERE ename LIKE '%s%'

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

SELECT \*

FROM emp

WHERE ename LIKE '\_\_A%'

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

SELECT \*

FROM emp

WHERE ename LIKE '%A\_\_'

=> display employees name contains 4 chars ?

SELECT \*

FROM emp

WHERE ename LIKE '\_\_\_\_'

=> display employees joined in jan month ?

YYYY-MM-DD

SELECT \*

FROM emp

WHERE hiredate LIKE '\_\_\_\_\_01\_\_\_'

=> display employees joined in 1981 year ?

SELECT \*

FROM emp

WHERE hiredate LIKE '1981%'

Question :-

1

SELECT \*

FROM emp

WHERE job IN ('CLERK','%MAN%')

=> above query returns

A ERROR

B CLERK,MANAGER rows

C only CLERK rows

D none

ans :- C

SELECT \*

FROM emp

WHERE job ='CLERK' OR job LIKE '%MAN%'

=> returns CLERK,MANAGER,SALESMAN

2

SELECT \*

FROM emp

WHERE sal BETWEEN 5000 AND 2000

a ERROR

b returns rows

c returns no rows

d none

ans :- c

SELECT \*

FROM emp

WHERE sal BETWEEN 2000 AND 5000 (sal>=2000 and sal<=5000)

SELECT \*

FROM emp

WHERE sal BETWEEN 5000 AND 2000 (sal>=5000 and sal<=2000)

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

upper and lower.

17-apr-21

IS operator :-

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

=> use IS operator for NULL comparision

WHERE COLNAME IS NULL

WHERE COLNAME IS NOT NULL

=> display employees not earning commission ?

SELECT \* FROM emp WHERE comm IS NULL

=> display employees earning commission ?

SELECT \* FROM emp WHERE comm IS NOT NULL

summary :-

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

WHERE COL BETWEEN V1 AND V2

WHERE COL LIKE 'PATTERN'

WHERE COL IS NULL / NOT NULL

Expressions :-

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

display employee names and annual salaries ?

SELECT ename,sal\*12

FROM emp

ename no column name

smith 9600

ALIAS :-

--------

=> alias means another name or alternative name.

=> used to change column heading.

COLNAME/EXPR [AS] ALIAS

SELECT ename,sal\*12 AS annsal

FROM emp

ename annsal

smith 9600

=> if alias contains space then enclose it in " "

SELECT ename,sal\*12 as "annual salary"

FROM emp

=> display ename,sal,hra,da,tax,totsal ?

hra = house rent allowance = 20% on sal

da = dearness allowance = 30% on sal

tax = 10% on sal

totsal = sal + hra + da - tax

SELECT ename,sal,

sal\*0.2 as hra,

sal\*0.3 as da,

sal\*0.1 as tax,

sal+(sal\*0.2)+(sal\*0.3)-(sal\*0.1) as totsal

FROM emp

ename sal hra da tax totsal

SMITH 800.00 160.00000 240.00000 80.00000 1120.00000

computed columns :-

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

=> computed column values are calculated from other columns.

CREATE TABLE emp\_salary

(

empno int,

ename varchar(10),

sal money,

hra as sal\*0.2,

da as sal\*0.3,

tax as sal\*0.1,

totsal as sal+hra+da-tax

)

INSERT INTO emp\_salary(empno,ename,sal) values(100,'A',5000)

select \* from emp\_salary

100 A 5000.00 1000.00000 1500.00000 500.00000 7000.00000

ORDER BY clause :-

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

=> ORDER BY clause is used to sort data based on one or more

columns either in ascending or in descending order.

SELECT columns

FROM tabname

[WHERE condition]

ORDER BY <col> ASC/DESC

=> default order is ascending for descending order use DESC option

=> arrange employee list name wise asc ?

SELECT \*

FROM emp

ORDER BY ename ASC

=> arrange employee list sal wise desc ?

SELECT \*

FROM emp

ORDER BY sal DESC

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

SELECT empno,ename,sal,deptno

FROM emp

ORDER BY deptno ASC,sal DESC

1 A 5000 10 4 D 6000 10

2 B 3000 20 1 A 5000 10

3 C 4000 30 ========> 6 F 2000 10

4 D 6000 10 5 E 7000 20

5 E 7000 20 2 B 3000 20

6 F 2000 10 3 C 4000 30

scenario :-

STUDENT

SNO SNAME M P C

1 A 80 90 70

2 B 60 50 70

3 C 90 70 80

4 D 90 80 70

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

SELECT \*

FROM student

ORDER BY (m+p+c)/3 DESC,m desc,p desc

4 D 90 80 70

3 C 90 70 80

1 A 80 90 70

2 B 60 50 70

NOTE :-

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

SELECT \*

FROM emp

ORDER BY 6 DESC

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

=> ORDER BY number should be according to select list but not

according to table.

=> arrange employee list sal wise desc ?

SELECT empno,ename,sal,deptno

FROM emp

ORDER BY 6 DESC => error

SELECT empno,ename,sal,deptno

FROM emp

ORDER BY 3 DESC

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

SELECT empno,ename,sal,deptno

FROM emp

ORDER BY 4 ASC,3 DESC

NOTE :- in sorting nulls treated low i.e. in ascending order nulls

displayed first and in desc order nulls displayed last.

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

DML commands :- (Data Manipulation Language)

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

INSERT

UPDATE

DELETE

MERGE

=> all DML commands acts on table data

=> in sql server by default every command is auto committed i.e. automatically saved.

=> to stop auto commit execute the following command.

SET IMPLICIT\_TRANSACTIONS ON

=> to save the operation execute COMMIT

=> to cancel the operation execute ROLLBACK

20-apr-21

UPDATE command :-

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

=> command used to modify data in a table.

UPDATE <tabname>

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

[WHERE condition]

=> update all employees comm to 500 ?

UPDATE emp SET comm=500

=> to save the above update execute commit

=> to cancel update execute rollback.

=> update employee sal to 2000 where empno=7369 ?

UPDATE emp SET sal=2000 WHERE empno=7369

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

UPDATE emp SET comm=500 WHERE comm IS NULL

=> update employee comm to null whose comm <> null ?

UPDATE emp SET comm = NULL WHERE comm IS NOT NULL

NULL assignment use =

NULL comparision use IS

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

as salesman and joined in 1981 year ?

UPDATE emp

SET sal = sal + (sal\*0.1) , comm = comm + (comm\*0.2)

WHERE job='SALESMAN'

AND

hiredate LIKE '1981%'

=> transfer all employees from 10th to 20th dept ?

UPDATE emp

SET deptno = 20

WHERE deptno = 10

DELETE command :-

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

=> command used to delete row/rows from table

=> we can delete all rows or specific rows

DELETE FROM <tabname> [WHERE condition]

=> delete all rows from emp table ?

DELETE FROM emp

=> delete employees who are joined in 1981 year ?

DELETE FROM emp WHERE hiredate LIKE '1981%'

creating new table from existing table (replica) :-

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

SELECT columns/\* INTO <new-tabname>

FROM <old-tabname>

WHERE condition

Examples :-

create replica of emp table ?

SELECT \* INTO emp10

FROM emp

=> above command creates new table with name EMP10 and copies

the rows and columns of emp to emp10.

copying specific rows and columns :-

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

SELECT empno,ename,job,sal INTO emp11

FROM emp

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

copying only structure (cols) but not data(rows) :-

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

SELECT \* INTO emp12

FROM emp

WHERE 1=2

copying data from one table to another table :-

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

INSERT INTO <TARGET-TABNAME>

SELECT COLUMNS/\* FROM <SOURCE-TABNAME>

copy data from emp to emp12 ?

INSERT INTO emp12

SELECT \* FROM emp

21-apr-21

MERGE command :-

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

=> command used to merge data into a table

=> command used to manage replicas

=> merge is the combination of insert & update

scenario :-

20/04/21

CREATE TABLE CUSTS

(

CID INT,

CNAME VARCHAR(10),

CADDR VARCHAR(10)

)

INSERT INTO CUSTS VALUES(1,'A','HYD'),(2,'B','MUM')

CUSTS

CID NAME ADDR

1 A HYD

2 B MUM

creating replica :-

SELECT \* INTO CUSTT FROM CUSTS

CUSTT

CID NAME ADDR

1 A HYD

2 B MUM

21/04/21

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

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

CUSTS

CID NAME ADDR

1 A BLR => updated

2 B MUM

3 C DEL => inserted

=> use MERGE command to apply changes made to CUSTS to CUSTT

MERGE INTO <TARGET-TABLE> <ALIAS>

USING <SOURCE-TABLE> <ALIAS>

ON (CONDITION)

WHEN MATCHED THEN

UPDATE

WHE NOT MATCHED THEN

INSERT

MERGE INTO CUSTT T

USING CUSTS S

ON (S.CID=T.CID)

WHEN MATCHED THEN

UPDATE SET T.ADDR = S.ADDR

WHEN NOT MATCHED THEN

INSERT VALUES(S.CID,S.NAME,S.ADDR);

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

DDL commands :- (Data Definition Language)

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

CREATE

ALTER

DROP

TRUNCATE

=> DDL commands acts on table structure (columns)

ALTER command :-

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

=> command used to moodify table structure

=> using ALTER we can

1 add columns

2 drop columns

3 modify column

Adding column :-

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

ALTER TABLE <tabname>

ADD colname datatype(size)

Ex :- add column gender to emp table ?

ALTER TABLE emp

ADD gender CHAR(1)

=> after adding by default the column is filled with NULLS , use

update command to insert data into new column.

UPDATE emp SET gender='M' WHERE empno=7369

Droping column :-

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

ALTER TABLE <tabname>

DROP COLUMN colname

=> drop column gender from emp ?

ALTER TABLE emp

DROP COLUMN gender

Modifying a column :-

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

1 changing field size

2 changing datatype

ALTER TABLE <TABNAME>

ALTER COLUMN COLNAME DATATYPE(SIZE)

=> increase size of ename to 20 ?

ALTER TABLE emp

ALTER COLUMN ename VARCHAR(20)

ALTER TABLE emp

ALTER COLUMN ename VARCHAR(10)

ALTER TABLE emp

ALTER COLUMN ename VARCHAR(8)

ALTER TABLE emp

ALTER COLUMN ename VARCHAR(5) => ERROR (name already contains 6 chars)

=> change datatype of deptno to tinyint ?

ALTER TABLE emp

ALTER COLUMN deptno TINYINT

ALTER TABLE emp

ALTER COLUMN empno TINYINT => ERROR because empnos are not

in TINYINT range.

22-apr-21

DROP command :-

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

=> command used to drop table from database.

=> drops table structure along with data.

syntax :- DROP TABLE <tabname>

Ex :- DROP TABLE emp

TRUNCATE command :-

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

=> command used to delete all the data from table but keeps structure

=> will empty the table

=> releases memory allocated for table

syn :- TRUNCATE TABLE <tabname>

Ex :- TRUNCATE TABLE emp

DROP VS DELETE VS TRUNCATE :-

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

DROP DELETE TRUNCATE

1 DDL DML DDL

2 drops structure deletes only deletes only data

with data data but not but not structure

structure

DELETE VS TRUNCATE :-

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

DELETE TRUNCATE

1 DML DDL

2 can delete specific deletes only all rows but

row/rows cannot delete specific row

3 where cond can be where cond cannot be used

used with delete with truncate

4 deletes row-by-row deletes all rows at a time

5 slower faster

6 will not release releases memory

memory

7 will not reset identity will reset identity

SP\_RENAME :- command used to change tablename and column name.

sp\_rename 'old name','new name' (sp => stored procedure)

=> rename table emp to employees ?

SP\_RENAME 'emp','employees'

=> rename column comm to bonus ?

SP\_RENAME 'employees.comm','bonus'

How to change column positions ?

STEP 1 :- create new table

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

from emp

STEP 2 :- drop original table

drop table emp

STEP 3 :- rename temp to emp

sp\_rename 'temp','emp'

23-apr-21

IDENTITY :-

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

=> used to generate sequence numbers.

=> used to auto increment column values.

syntax :- IDENTITY(SEED,INCR)

SEED => start

optional

default 1

INCR => increment

optional

default 1

Example :-

CREATE TABLE cust

(

cid int IDENTITY(100,1),

cname varchar(10)

)

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

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

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

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

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

SELECT \* FROM cust

CID CNAME

100 A

101 B

102 C

103 D

104 E

NOTE :- delete will not reset identity but truncate will reset identity

DELETE FROM CUST

INSERT INTO CUST(CNAME) VALUES('K')

SELECT \* FROM CUST

CID CNAME

105 K

TRUNCATE TABLE CUST

INSERT INTO CUST(CNAME) VALUES('X')

SELECT \* FROM CUST

CID CNAME

100 X

How to reset identity manually :-

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

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

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

DBCC => db consistency check

How to insert explicit value into identity column :-

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

=> by default sql server will not allow explicit value into

identity column. To insert explicit value into identity column execute the

following command

INSERT INTO CUST(CID,CNAME) VALUES(110,'G') => ERROR

SET IDENTITY\_INSERT CUST ON

INSERT INTO CUST(CID,CNAME) VALUES(110,'G') => 1 row affected

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

24-APR-21

Built-in Functions in SQL SERVER :-

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

=> a function accepts some input performs some calculation and

returns one value.

Types of functions :-

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

1 date

2 string

3 mathematical

4 conversion

5 special

6 analytical

7 aggregate

Date functions :-

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

1 GETDATE() :- returns current date & time

SELECT GETDATE() => 2021-04-24 08:15:27.853

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

syn :- DATEPART(interval,date)

DATEPART(yy,GETDATE()) => 2021

DATEPART(mm,GETDATE()) => 04

DATEPART(dd,GETDATE()) => 24

DATEPART(dw,GETDATE()) => 07 (dw => day of the week)

01 SUNDAY

02 MONDAY

07 SATURDAY

DATEPART(q,GETDATE()) => 02 (Q => Quarter)

01 JAN-MAR

02 APR-JUN

03 JUL-SEP

04 OCT-DEC

DATEPART(hh,GETDATE()) => 8

DATEPART(mi,GETDATE()) => 22

DATEPART(ss,GETDATE()) => 20

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

SELECT \* FROM emp

WHERE DATEPART(yy,hiredate) IN (1980,1983,1985)

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

SELECT \* FROM emp

WHERE DATEPART(mm,hiredate) IN (01,04,12)

=> display employees joined on sunday ?

SELECT \* FROM emp

WHERE DATEPART(dw,hiredate)=1

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

SELECT \* FROM emp

WHERE DATEPART(yy,hiredate)=1981

AND

DATEPART(q,hiredate)=2

DATENAME() :-

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

MM DW

DATEPART 04 07

DATENAME APRIL SATURDAY

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

SELECT DATENAME(dw,'1947-08-15')

=> write a query to print smith joined on wednesday

allen joined on friday ?

SELECT ename + ' joined on ' + datename(dw,hiredate)

FROM emp

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

DATEDIFF(interval,start date,end date)

SELECT DATEDIFF(yy,'2020-04-24',GETDATE()) => 1

SELECT DATEDIFF(mm,'2020-04-24',GETDATE()) => 12

SELECT DATEDIFF(dd,'2020-04-24',GETDATE()) => 365

DISPLAY ename , experience ?

SELECT ename,DATEDIFF(yy,hiredate,GETDATE()) as experience

FROM emp

DISPLAY ename,experience ?

M years N months

Experience = 40 months = 3 years 4 months

years = months/12 = 40/12 = 3

months = months%12 = 40%12 = 4

SELECT ename,

DATEDIFF(mm,hiredate,GETDATE())/12 as years,

DATEDIFF(mm,hiredate,GETDATE())%12 as months

FROM emp

DATEADD() :-

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

DATEADD(interval,int,date)

SELECT DATEADD(dd,10,GETDATE()) => 2021-05-04

SELECT DATEADD(mm,1,GETDATE()) => 2021-05-24

SELECT DATEADD(yy,1,GETDATE()) => 2022-04-24

SELECT DATEADD(mm,-1,GETDATE()) => 2021-03-24

scenario :-

display ename hiredate dor ?

dor = date or resign = hiredate + 40 years

SELECT ename,hiredate,

DATEADD(yy,40,hiredate) as dor

FROM emp

scenario :-

GOLD\_RATES

DATEID RATE

2015-01-01 ?

2021-04-26 ?

=> display today's gold rate ?

SELECT \* FROM gold\_rates WHERE dateid=GETDATE()

=> display yesterday's gold rate ?

SELECT \* FROM gold\_rates WHERE dateid = DATEADD(dd,-1,GETDATE())

=> display last month same day gold rate ?

SELECT \* FROM gold\_rates WHERE dateid = DATEADD(mm,-1,GETDATE())

=> display last year same day gold rate ?

SELECT \* FROM gold\_rates WHERE dateid = DATEADD(yy,-1,GETDATE())

EOMONTH() :- returns last day of the month

EOMONTH(DATE,INT)

SELECT EOMONTH(GETDATE(),0) => 2021-04-30

SELECT EOMONTH(GETDATE(),1) => 2021-05-31

SELECT EOMONTH(GETDATE(),-1) => 2021-03-31

=> display current month 1st day ?

=> display next month 1st day ?

=> display current year 1st day ?

=> display next year 1st day ?

STRING functions :-

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

1 UPPER() :- converts string to uppercase

UPPER(string)

SELECT UPPER('hello') => HELLO

2 LOWER() :- converts string to lowercase

LOWER(string)

SELECT LOWER('HELLO') => hello

display EMPNO,ENAME,SAL ? display names in lowercase ?

SELECT empno,LOWER(ename),sal FROM emp

=> convert names to lowercase in table ?

UPDATE emp SET ename=LOWER(ename)

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

LEN(string)

SELECT LEN('hello') => 5

display employees name contains 5 chars ?

SELECT \* FROM emp WHERE ename LIKE '\_\_\_\_\_'

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

LEFT() :- returns given no of chars from left side

LEFT(string,int)

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

=> display employees name starts with 's' ?

SELECT \* FROM emp WHERE ename LIKE 's%'

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

scenario :-

generate emailids for employees ?

empno ename emailid

7369 smith smi736@microsoft.com

7499 allen all749@microsoft.com

SELECT empno,ename,

LEFT(ename,3) + LEFT(empno,3) + '@microsoft.com' as emailid

FROM emp

=> store emailids in db ?

1 add emailid column to emp table

ALTER TABLE emp

ADD emailid VARCHAR(30)

2 update column with emailids ?

UPDATE emp

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

RIGHT() :- returns given no of chars from right side.

RIGHT(string,int)

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

=> display employees name ends with 's' ?

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

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

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

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

or

ename like 'b%b'

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

SUBSTRING() :-

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

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

SUBSTRING(string,start,len)

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

CHARINDEX() :- returns position of the char in a string

CHARINDEX(char,string,[start])

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

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

27-apr-21

REPLICATE() :-

=> used to repeat the string for given no of times

REPLICATE(string,int)

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

display ename sal ?

\*\*\*\*

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

scenario :-

ACCOUNTS

ACCNO

12345678967

=> your a/c no XXXX8967 debited ????

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

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

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

REPLACE(str1,str2,str3)

SELECT REPLACE('hello','ell','abc') => habco

SELECT REPLACE('hello','l','abc') => heabcabco

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

STUFF() :- used to replace one string with another based on length

STUFF(string,start,len,string)

SELECT STUFF('hello welcome',7,4,'abc') => hello abcome

Mathematical functions :-

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

1 ABS() :- returns absolute value

SELECT abs(-10) => 10

2 POWER() :- used to calculate power

SELECT POWER(3,2) => 9

3 SQRT() :- returns square root

SELECT SQRT(16) => 4

4 SQUARE() :- returns square

SELECT SQUARE(5) => 25

5 SIGN() :- returns given expr is positive or negative

SELECT SIGN(10) => 1

SELECT SIGN(-10) => -1

SELECT SIGN(0) => 0

6 ROUND :- used to round number to integer or decimal places based

on the average.

ROUND(number,decimal places,[optional])

SELECT ROUND(123.456,0) => 123

123-----------------123.5------------------124

number >= avg => rounded to highest

number < avg => rounded to lowest

SELECT ROUND(123.5,0) => 124

SELECT ROUND(123.4567,2) => 123.46

SELECT ROUND(123.4537,2) => 123.45

SELECT ROUND(123.4567,3) => 123.457

SELECT ROUND(386,-2) => 400

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

SELECT ROUND(386,-1) => 390

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

SELECT ROUND(386,-3) => 0

0-----------------------500-----------------------100

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

4570 4600 5000

NOTE :-

ROUND(number,decimal places,[optional])

optional => 0 => acts like round (default)

optional => 1 => acts like trunc

=> round acts according to avg but trunc always rounds to lowest

SELECT ROUND(358,-2,1) => 300

SELECT ROUND(358.456,2,1) => 358.45

CEILING() :- rounds numbers always to highest

CEILING(number)

SELECT CEILING(3.1) => 4

FLOOR() :- rounds number always to lowest

FLOOR(3.1) => 3

conversion functions :-

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

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

=> SQL SERVER provided two conversion functions

1 CAST

2 CONVERT

CAST :-

CAST(source-expr as target-type)

SELECT CAST(3.9 AS INT) => 3

display smith earns 800

allen earns 1600 ?

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

SELECT ename + ' earns ' + CAST(sal as varchar) FROM emp

display smith joined on 1980-12-17 ?

SELECT ename + ' joined on ' + CAST(hiredate as varchar) FROM emp

28-apr-21

CONVERT() :-

CONVERT(target-type,source-expr)

SELECT CONVERT(int,30.15) => 30

diff b/w cast & convert ?

=> using convert function we can display dates & numbers in

different formats and that is not possible with cast function.

displaying dates in different formats :-

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

CONVERT(varchar,date,style-number)

Without century With century (yyyy) Standard Input/Output (3)

1 101 U.S. 1 = mm/dd/yy

101 = mm/dd/yyyy

2 102 ANSI 2 = yy.mm.dd

102 = yyyy.mm.dd

3 103 British/French 3 = dd/mm/yy

103 = dd/mm/yyyy

4 104 German 4 = dd.mm.yy

104 = dd.mm.yyyy

5 105 Italian 5 = dd-mm-yy

105 = dd-mm-yyyy

6 106 - 6 = dd mon yy

106 = dd mon yyyy

7 107 7 = Mon dd, yy

107 = Mon dd, yyyy

8 108 - hh:mi:ss

9 109 Default + milliseconds mon dd yyyy hh:mi:ss:mmmAM (or PM)

10 110 USA 10 = mm-dd-yy

110 = mm-dd-yyyy

11 111 JAPAN 11 = yy/mm/dd

111 = yyyy/mm/dd

12 112 ISO 12 = yymmdd

112 = yyyymmdd

- 13 or 113 (1,2) Europe default + milliseconds dd mon yyyy hh:mi:ss:mmm (24h)

14 114 - hh:mi:ss:mmm (24h)

- 20 or 120 (2) ODBC canonical yyyy-mm-dd hh:mi:ss (24h)

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

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

- 23 ISO8601 yyyy-mm-dd

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

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

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

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

- 130 (1,2) Hijri (5) dd mon yyyy hh:mi:ss:mmmAM

In this style, mon represents a multi-token Hijri unicode representation of the full month name. This value does not render correctly on a default US installation of SSMS.

- 131 (2) Hijri (5) dd/mm/yyyy hh:mi:ss:mmmAM

display empno,ename,hiredate ? display hiredates in mm/dd/yyyy format ?

SELECT empno,ename,

CONVERT(varchar,hiredate,101) as hiredate

from emp

MONEY AND SMALLMONEY STYLES :-

Value Output

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

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

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

display empno,ename,sal ? display salaries with thousand seperator ?

SELECT empno,ename,CONVERT(varchar,sal,1) as sal

from emp

7369 smith 800.00

7499 allen 1,600.00

7521 ward 1,250.00

7566 jones 2,975.00

special functions :-

ISNULL() :- used to convert null values.

ISNULL(arg1,arg2)

if arg1 = null returns arg2

if arg1 <> null returns arg1 only

SELECT ISNULL(100,200) => 100

SELECT ISNULL(NULL,200) => 200

Display empno,ename,sal,comm,totsal ?

SELECT empno,ename,sal,comm,sal+comm as totsal

FROM emp

7369 smith 800.00 NULL NULL

7499 allen 1600.00 300.00 1900.00

7521 ward 1250.00 500.00 1750.00

7566 jones 2975.00 NULL NULL

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

FROM emp

7369 smith 800.00 NULL 800.00

7499 allen 1600.00 300.00 1900.00

7521 ward 1250.00 500.00 1750.00

7566 jones 2975.00 NULL 2975.00

display empno,ename,sal,comm ? if comm = NULL display N/A ?

SELECT empno,ename,sal,ISNULL(comm,'N/A') as comm

FROM emp => ERROR

SELECT empno,ename,sal,

ISNULL(CAST(comm as varchar),'N/A') as comm

FROM emp

ISNUMERIC(expr) :-

if expr is numeric then returns 1

if expr is not numeric then returns 0

SELECT ISNUMERIC(100) => 1

SELECT ISNUMERIC('ABC') => 0

ISDATE(expr) :-

if expr is date returns 1

if expr is not date returns 0

SELECT ISDATE(GETDATE()) => 1

SELECT ISDATE(100) => 0

29-APR-21

Analytical functions :-

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

RANK , DENSE\_RANK :-

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

=> both functions are used to calculate ranks

=> ranking is based on some column

=> for ranks functions data must be sorted

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

DENSE\_RANK() OVER (ORDER BY COLNAME ASC/DESC)

=> display employee ranks based on sal and highest paid employee

should get 1st rank ?

SELECT empno,ename,sal,

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

FROM EMP

SELECT empno,ename,sal,

DENSE\_RANK() OVER (ORDER BY sal DESC) AS RNK

FROM EMP

=> diff b/w rank & dense\_rank ?

=> rank function generates gaps but dense\_rank will not generate gaps.

=> in rank function ranks may not be in sequence but in dense\_rank

function ranks will be always in sequence.

SAL RNK DRNK

5000 1 1

4000 2 2

3000 3 3

3000 3 3

3000 3 3

2000 6 4

2000 6 4

1000 8 5

=> display ranks of the employees based on sal , if salaries are

same then ranking should be based on experience ?

SELECT empno,ename,hiredate,sal,

DENSE\_RANK() OVER (ORDER BY sal DESC,hiredate ASC) AS RNK

FROM EMP

7839 king 1981-11-17 5000.00 1

7902 ford 1981-12-03 3000.00 2

7788 scott 1982-12-09 3000.00 3

7566 jones 1981-04-02 2975.00 4

7698 blake 1981-05-01 2850.00 5

7782 clark 1981-06-09 2450.00 6

7499 allen 1981-02-20 1600.00 7

7844 turner 1981-09-08 1500.00 8

7934 miller 1982-01-23 1300.00 9

7521 ward 1981-02-22 1250.00 10

7654 martin 1981-09-28 1250.00 11

PARTITION BY clause :-

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

=> used to find ranks with in group for example to find ranks

with in dept first we need to divide the table dept wise and

apply dense\_rank function on each dept.

=> display ranks of the employees with in dept and highest

paid employee should get 1st rank ?

SELECT empno,ename,sal,deptno,

DENSE\_RANK() OVER (PARTITION BY DEPTNO

ORDER BY sal DESC) AS rnk

FROM emp

7839 king 5000.00 10 1

7782 clark 2450.00 10 2

7934 miller 1300.00 10 3

7902 ford 3000.00 20 1

7788 scott 3000.00 20 1

7566 jones 2975.00 20 2

7876 adams 1100.00 20 3

7369 smith 800.00 20 4

7698 blake 2850.00 30 1

7499 allen 1600.00 30 2

7844 turner 1500.00 30 3

7521 ward 1250.00 30 4

7654 martin 1250.00 30 4

7900 james 950.00 30 5

ROW\_NUMBER() :-

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

=> returns record numbers after sorting

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

SELECT empno,ename,sal,

ROW\_NUMBER() OVER (ORDER BY empno ASC) as rno

FROM emp

7369 smith 800.00 1

7499 allen 1600.00 2

7521 ward 1250.00 3

7566 jones 2975.00 4

7654 martin 1250.00 5

7698 blake 2850.00 6

7782 clark 2450.00 7

7788 scott 3000.00 8

7839 king 5000.00 9

7844 turner 1500.00 10

7876 adams 1100.00 11

7900 james 950.00 12

7902 ford 3000.00 13

7934 miller 1300.00 14

AGGREGATE FUNCTIONS / GROUP FUNCTIONS :-

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

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

MAX() :- returns maximum value

MAX(arg)

1 SELECT MAX(sal) FROM emp => 5000

2 SELECT MAX(hiredate) FROM emp => 1983-01-12

MIN() :- returns mininum value

MIN(arg)

SELECT MIN(sal) FROM emp => 800

SUM() :- returns total

SUM(arg)

SELECT SUM(sal) FROM emp => 29025

=> round total sal to hundreds ?

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

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

=> after rounding display total sal with thousand seperator ?

SELECT CONVERT(VARCHAR,ROUND(SUM(sal),-2),1)

FROM emp

output :- 29,000.00

display running total ?

SELECT empno,ename,sal,

SUM(sal) OVER (ORDER BY empno ASC) as running\_total

FROM emp

7369 smith 800.00 800

7499 allen 1600.00 2400

7521 ward 1250.00 3650.00

7566 jones 2975.00 6625.00

AVG() :- returns average

AVG(arg)

SELECT AVG(sal) FROM emp => 2073.2142

=> round avg(sal) to integer ?

SELECT CAST(AVG(SAL) AS INT) FROM EMP => 2073

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

COUNT(arg)

SELECT COUNT(empno) FROM emp => 14

SELECT COUNT(comm) FROM emp => 4

SELECT COUNT(deptno) FROM emp => 14

SELECT COUNT(DISTINCT deptno) FROM emp => 3

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

SELECT COUNT(\*) FROM emp => 14

T1

F1

10

NULL

20

NULL

30

COUNT(F1) = 3 => excludes nulls

COUNT(\*) = 5 => includes nulls

30-APR-21

=> display no of employees joined in 1981 year ?

SELECT COUNT(\*)

FROM emp

WHERE DATEPART(yy,hiredate)=1981

=> display no of employees joined on sunday ?

SELECT COUNT(\*)

FROM emp

WHERE DATENAME(dw,hiredate)='sunday'

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

SELECT COUNT(\*)

FROM emp

WHERE DATEPART(yy,hiredate) = 1981

and

DATEPART(q,hiredate) = 2

NOTE :-

1

aggregate functions are not allowed in where clause they

are allowed only in select,having clauses.

SELECT ename

FROM emp

WHERE sal = MAX(sal) => ERROR

=> to overcome this problem use subquery.

2 SELECT col1,col2 FROM tabname

Rule :- no of values return by col1 must be equal to no of

values return by col2.

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

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

14 1

2 SELECT ename,ROUND(sal,-1) FROM emp => EXECUTED

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

14 14

3 SELECT ROUND(sal,-1),MAX(sal) FROM emp => ERROR

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

14 1

4 SELECT MIN(sal),MAX(sal) FROM EMP => EXECUTED

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

1 1

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

CASE statement :-

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

=> case statement is similar to IF-THEN-ELSE.

=> case statement is similar to switch case.

=> case statements are 2 types

1 simple case

2 searched case

simple case :-

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

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

CASE expr

WHEN value1 THEN return expr1

WHEN value2 THEN return expr2

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

ELSE return expr

END

=> display ename,job ?

if job=clerk display worker

job=manager boss

job=president big boss

others 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

=> increment employee salaries as follows ?

if deptno=10 incr sal by 10%

20 15%

30 20%

others 5%

UPDATE emp

SET sal = CASE deptno

WHEN 10 THEN sal+(sal\*0.1)

WHEN 20 THEN sal+(sal\*0.15)

WHEN 30 THEN sal+(sal\*0.2)

ELSE sal+(sal\*0.05)

END

searched case :-

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

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

CASE

WHEN cond1 THEN return expr1

WHEN cond2 THEN return expr2

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

ELSE return expr

END

display ename,sal,salrange ?

if sal<3000 display losal

sal>3000 display hisal

sal=3000 avgsal

SELECT ename,sal,

CASE

WHEN sal>3000 THEN 'Hisal'

WHEN sal<3000 THEN 'Losal'

ELSE 'Avgsal'

END as salrange

FROM emp

STUDENT

sno sname s1 s2 s3

1 A 80 90 70

2 B 30 60 50

=> display sno,total,avg,result ?

SELECT sno,s1+s2+s3 as stotal,

(s1+s2+s3)/3 as savg,

case

when s1>=35 and s2>=35 and s3>=35 and (s1+s2+s3)/3 >=70 THEN 'Distinction'

when s1>=35 and s2>=35 and s3>=35 and (s1+s2+s3)/3 >=60 THEN 'First'

when s1>=35 and s2>=35 and s3>=35 and (s1+s2+s3)/3 >=50 THEN 'Second'

when s1>=35 and s2>=35 and s3>=35 and (s1+s2+s3)/3 >=35 THEN 'Third'

else 'fail'

end as result

FROM student

01-may-21

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.

EMP

EMPNO ENAME SAL DEPTNO

1 A 5000 10

2 B 3000 20 group by 10 12000

3 C 4000 30 ----------> 20 9000

4 D 6000 20 30 4000

5 E 7000 10

detailed data summarized data

=> using GROUP BY clause we can convert detailed data into

summarized data which is useful for analysis.

Syntax :-

SELECT columns

FROM tabname

[WHERE condition]

GROUP BY <col>

[HAVING <condition>]

[ORDER BY <col> ASC/DESC]

Execution :-

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

FROM

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

Examples :-

=> display dept wise total sal ?

SELECT deptno,SUM(sal) as totsal

FROM emp

GROUP BY deptno

Execution :-

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

FROM emp :-

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

EMPNO ENAME SAL DEPTNO

1 A 5000 10

2 B 3000 20

3 C 4000 30

4 D 6000 20

5 E 7000 10

GROUP BY deptno :-

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

10

1 A 5000

5 E 7000

20

2 B 3000

4 D 6000

30

3 C 4000

SELECT deptno,SUM(sal) :-

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

10 12000

20 9000

30 4000

=> display job wise no of employees ?

SELECT job,COUNT(\*) as cnt

FROM emp

GROUP BY job

=> display year wise no of employees joined ?

SELECT DATEPART(yy,hiredate) as year,COUNT(\*)

FROM emp

GROUP BY DATEPART(yy,hiredate)

SELECT DATEPART(yy,hiredate) as year,COUNT(\*)

FROM emp

GROUP BY year => ERROR

NOTE :- column alias cannot be referenced in GROUP BY clause

because GROUP BY clause is executed before SELECT.

=> display departments where more than 3 employees working ?

SELECT deptno,COUNT(\*)

FROM emp

WHERE COUNT(\*) > 3

GROUP BY deptno => ERROR

=> SQL SERVER cannot calculate dept wise count before group by

it can calculate only after group by , so apply the condition

COUNT(\*) > 3 after group by using HAVING clause.

SELECT deptno,COUNT(\*) as cnt

FROM emp

GROUP BY deptno

HAVING count(\*) > 3

03-may-21 :-

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

no of employees > 3 ?

SELECT deptno,COUNT(\*)

FROM emp

WHERE deptno IN (10,20)

GROUP BY deptno

HAVING COUNT(\*) > 3

FROM emp :-

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

EMPNO ENAME SAL DEPTNO

1 A 5000 10

2 B 3000 20

3 C 4000 30

4 D 6000 20

5 E 7000 10

6 F 4000 20

7 G 3000 20

WHERE deptno IN (10,20) :-

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

EMPNO ENAME SAL DEPTNO

1 A 5000 10

2 B 3000 20

4 D 6000 20

5 E 7000 10

6 F 4000 20

7 G 3000 20

GROUP BY deptno :-

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

10

1 A 5000

5 E 7000

20

2 B 3000

4 D 6000

6 F 4000

7 G 3000

HAVING COUNT(\*) > 3 :-

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

20

2 B 3000

4 D 6000

6 F 4000

7 G 3000

SELECT deptno,COUNT(\*) :-

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

20 4

=> diff b/w WHERE and HAVING ?

WHERE HAVING

1 filters rows filters groups

2 conditions applied conditions applied

before group by after group by

3 use where clause if use having clause

condition doesn't contain if condition contains

aggregate function aggregate function

=> display dept wise total sal where deptno=10,20 and

total sal > 10000 ?

SELECT deptno,SUM(sal)

FROM emp

WHERE deptno IN (10,20)

GROUP BY deptno

HAVING SUM(sal) > 10000

Exercise :-

PERSONS

NAME AGE GENDER ADDR CITY STATE

1 display state wise population ?

2 display gender wise population ?

3 display age group wise population ?

4 display state wise population for the states=AP,TS,KA,MH,UP

and total population > 5 crores ?

SELECT state,count(\*)

FROM persons

WHERE state IN ('AP','TS','KA','MH','UP')

GROUP BY state

HAVING COUNT(\*) > 50000000

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

SELECT deptno,job,COUNT(\*) as cnt

FROM emp

GROUP BY deptno,job

ORDER BY deptno ASC

EMP :-

10 MANAGER 7782 clark

10 PRESIDENT 7839 king

10 CLERK 7934 miller

20 ANALYST 7902 ford

20 CLERK 7876 adams

20 ANALYST 7788 scott

20 CLERK 7369 smith

20 MANAGER 7566 jones

30 SALESMAN 7654 martin

30 MANAGER 7698 blake

30 SALESMAN 7499 allen

30 SALESMAN 7521 ward

30 SALESMAN 7844 turner

30 CLERK 7900 james

output :-

10 clerk 1

10 manager 1

10 president 1

20 analyst 2

20 clerk 2

20 manager 1

30 clerk 1

30 manager 1

30 salesman 4

=> 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

ROLLUP and CUBE :-

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

=> both functions are used to calcualte 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 ISNULL(deptno,99) ASC

10 CLERK 1430.00

10 MANAGER 2695.00

10 PRESIDENT 5500.00

10 NULL 9625.00 => subtotal

20 ANALYST 6900.00

20 CLERK 2185.00

20 MANAGER 3421.25

20 NULL 12506.25 => subtotal

30 CLERK 1140.00

30 MANAGER 3420.00

30 SALESMAN 6720.00

30 NULL 11280.00 => subtotal

NULL NULL 33411.25 => grand total

CUBE :-

--------

=> CUBE displays subtotals for each group by column (deptno,job)

and also displays grand total.

SELECT deptno,job,SUM(sal) as totsal

FROM emp

GROUP BY CUBE(deptno,job)

ORDER BY ISNULL(deptno,99) asc , ISNULL(job,'ZZZ') asc

10 CLERK 1430.00

10 MANAGER 2695.00

10 PRESIDENT 5500.00

10 NULL 9625.00 => dept subtotal

20 ANALYST 6900.00

20 CLERK 2185.00

20 MANAGER 3421.25

20 NULL 12506.25 => dept subtotal

30 CLERK 1140.00

30 MANAGER 3420.00

30 SALESMAN 6720.00

30 NULL 11280.00 => dept subtotal

NULL ANALYST 6900.00 => job subtotal

NULL CLERK 4755.00 => job subtotal

NULL MANAGER 9536.25 => job subtotal

NULL PRESIDENT 5500.00 => job subtotal

NULL SALESMAN 6720.00 => job subtotal

NULL NULL 33411.25 => grand total

Exercise :-

1

SALES

DATEID PRODID CUSTID QTY AMOUNT

2021-05-04 100 10 1 5000

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

and display year wise subtotals ?

2018 1 ?

2 ?

3 ?

4 ?

?

2019 1 ?

2 ?

?

2 PERSONS

NAME AGE GENDER ADDR CITY STATE

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

and display state wise and gender wise subtotals ?

GROUPING\_ID() :-

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

=> this functions accepts group by columns and returns subtotals

belongs to which group by column.

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'

ELSE ' '

END AS subtotals

FROM emp

GROUP BY CUBE(deptno,job)

ORDER BY ISNULL(deptno,99) asc , ISNULL(job,'ZZZ') asc

10 CLERK 1430.00

10 MANAGER 2695.00

10 PRESIDENT 5500.00

10 NULL 9625.00 Dept Subtotal

20 ANALYST 6900.00

20 CLERK 2185.00

20 MANAGER 3421.25

20 NULL 12506.25 Dept Subtotal

30 CLERK 1140.00

30 MANAGER 3420.00

30 SALESMAN 6720.00

30 NULL 11280.00 Dept Subtotal

NULL ANALYST 6900.00 Job Subtotal

NULL CLERK 4755.00 Job Subtotal

NULL MANAGER 9536.25 Job Subtotal

NULL PRESIDENT 5500.00 Job Subtotal

NULL SALESMAN 6720.00 Job Subtotal

NULL NULL 33411.25 Grand Total

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

05-MAY-21 Integrity Constraints

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

=> Integrity Constraints are rules to maintain Data Integrity

i.e. Data quality or Data consistency.

=> Integrity Constraints prevents users from entering invalid data.

=> Integrity Constraints are used to enforce rules.

Types of Integrity Constraints :-

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

1 NOT NULL

2 UNIQUE

3 PRIMARY KEY

4 CHECK

5 FOREIGN KEY

6 DEFAULT

=> above constraints can be declared in two ways

1 column level

2 table level

column level :-

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

=> if constraints are declared immediately after declaring column

then it is called column level.

CREATE TABLE <tabname>

(

COLNAME DATATYPE(SIZE) CONSTRAINT,

COLNAME DATATYPE(SIZE) CONSTRAINT,

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

)

NOT NULL :-

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

=> NOT NULL constraint doesn't accept nulls.

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

CREATE TABLE emp20

(

empno int,

ename varchar(10) NOT NULL

)

INSERT INTO emp20 VALUES(100,'A')

INSERT INTO emp20 VALUES(101,NULL) => ERROR

UNIQUE :-

----------

=> UNIQUE constraint doesn't accept duplicates.

CREATE TABLE emp21

(

empno int,

emailid varchar(10) UNIQUE

)

INSERT INTO emp21 VALUES(100,'abc@gmail.com')

INSERT INTO emp21 VALUES(101,'abc@gmail.com') => ERROR

INSERT INTO emp21 VALUES(102,NULL) => accepted

INSERT INTO emp21 VALUES(103,NULL) => ERROR

NOTE :- unique constraint allows one NULL.

PRIMARY KEY :-

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

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

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

PRIMARY KEY = UNIQUE + NOT NULL

CREATE TABLE emp22

(

empno int PRIMARY KEY,

ename varchar(10)

)

INSERT INTO emp22 VALUES(100,'A')

INSERT INTO emp22 VALUES(100,'B') => ERROR

INSERT INTO emp22 VALUES(NULL,'B') => ERROR

=> because it doesn't allow duplicates and nulls so using

PRIMARY KEY column we can uniquely identify records in a table.

=> only one primary key is allowed per table. if we want two

primary keys then declare one column with primary key and

another column with UNIQUE and NOT NULL.

CREATE TABLE CUST

(

ACCNO INT PRIMARY KEY,

NAME VARCHAR(10),

AADHARNO INT UNIQUE NOT NULL

)

Diff b/w PRIMARY KEY & UNIQUE ?

PRIMARY KEY UNIQUE

1 doesn't allows nulls allows one null

2 only one primary key a table can have multiple unique

is allowed per table constraints

Diff b/w PRIMARY KEY & UNIQUE NOT NULL ?

PRIMARY KEY UNIQUE NOT NULL

1 a table allows only one pk a table can have multiple unique & not null.

06-may-21

CHECK constraint :-

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

=> use check constraint when rule based on condition.

syn :- CHECK(condition)

Example 1 :- sal must be min 3000 ?

CREATE TABLE emp23

(

empno int,

sal money CHECK(sal>=3000)

)

INSERT INTO emp23 VALUES(100,5000)

INSERT INTO emp23 VALUES(101,1000) => ERROR

INSERT INTO emp23 VALUES(102,NULL) => accepted

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

gender char(1) check(gender IN ('m','f'))

Example 3 :- amt must be multiple of 100 ?

amt money check(amt%100=0)

Example 4 :- pwd must be min 8 chars ?

pwd varchar(20) check(LEN(pwd)>=8)

Example 5 :- email should contain @

should end with .com,.co,.in

email varchar(30) check(email like '%@%'

and

(email like '%.com'

or

email like '%.co'

or

email like '%.in'

)

)

FOREIGN KEY :-

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

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

=> to establish relationship take primary key of one table and add

it to another table as foreign key and declare with references

constraint.

Example :-

DEPT

DNO DNAME LOC

10 HR BLR

20 IT HYD

EMP

EMPID ENAME SAL DNO REFERENCES DEPT(DNO)

1 A 5000 10

2 B 6000 20

3 C 7000 90 => NOT ACCEPTED

4 D 4000 10 => ACCEPTED

5 E 3000 NULL => ACCEPTED

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

entered in primary key column.

=> foreign key allows duplicates and nulls.

=> after declaring foreign key a relationship is established between

two tables and that relationship is called parent/child relationship

or master/detail relationship.

=> PK table is called parent and FK table is called child.

Example :-

----------

CREATE TABLE dept55

(

dno int PRIMARY KEY,

dname varchar(10) UNIQUE NOT NULL,

loc varchar(10)

)

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

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',5000,10)

INSERT INTO emp55 VALUES(2,'B',4000,90) => ERROR

INSERT INTO emp55 VALUES(3,'C',6000,10) => ACCEPTED

INSERT INTO emp55 VALUES(4,'D',3000,NULL) => ACCEPTED

Exercise :-

-----------

ACCOUNTS

ACCNO NAME ACTYPE BAL

rules :-

1 accno should not be duplicate & null

2 name should not be null

3 actype must be 'S','C','R'

4 bal must be min 1000

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

rules :-

1 trid must be automatically generated

2 ttype must be 'w' or 'd'

3 tdate must be current date

4 tamt must be multiple of 100

5 accno should match with accounts table accno.

07-may-21

DEFAULT :-

-----------

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

hiredate date default getdate()

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

default value.

CREATE TABLE emp24

(

empno int,

hiredate date default getdate()

)

INSERT INTO emp24(empno) VALUES(100)

INSERT INTO emp24 VALUES(101,'2021-01-01')

INSERT INTO emp24 VALUES(102,NULL)

SELECT \* FROM emp24

empno hiredate

100 2021-05-07

101 2021-01-01

102 NULL

TABLE LEVEL :-

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

=> if constraints are declared after declaring all columns then it

is called table level.

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

combination of columns.

CREATE TABLE <TABNAME>

(

COL1 DATATYPE(SIZE),

COL2 DATATYPE(SIZE),

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

CONSTRAINT(COL1,COL2,---)

)

Declaring check constraint at table level :-

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

MEDICINES

ID NAME MFD\_DT EXP\_DT

1 XXXX 2021-05-07 2021-01-01

RULE :- EXP\_DT > MFD\_DT

CREATE TABLE MEDICINES

(

ID INT PRIMARY KEY IDENTITY,

NAME VARCHAR(10),

MFD\_DT DATE,

EXP\_DT DATE ,

CHECK(EXP\_DT > MFD\_DT)

)

INSERT INTO MEDICINES(NAME,MFD\_DT,EXP\_DT)

VALUES('ABC',GETDATE(),'2021-01-01') => ERROR

INSERT INTO MEDICINES(NAME,MFD\_DT,EXP\_DT)

VALUES('ABC','2021-01-01',GETDATE()) => 1 row affected

composite primary key :-

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

=> if primary key constraint is declared for combination of columns

then it is called composite primary key.

=> when we are not able to uniquely identify records using single

column then use combinatination of columns to uniquely identify.

if combination of columns uniquely identifies then declare that

combination as primary key called composite primary key.

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

=> composite primary key is declared at table level.

Example :-

STUDENT COURSE

SID SNAME CID CNAME

1 A 10 SQL

2 B 11 .NET

REGISTRATIONS

SID CID DOR FEE

1 10 ?? 1000

1 11 ?? 1000

2 10 ?? 1000

=> in the above example SID,CID combination uniquelyu identifies

records , so declare this combination as primary key at table level.

CREATE TABLE STUDENT

(

SID INT PRIMARY KEY,

SNAME VARCHAR(10)

)

INSERT INTO STUDENT VALUES(1,'A'),(2,'B')

CREATE TABLE COURSE

(

CID INT PRIMARY KEY,

CNAME VARCHAR(10)

)

INSERT INTO COURSE VALUES(10,'SQL'),(11,'.NET')

CREATE TABLE REGISTRATIONS

(

SID INT REFERENCES STUDENT(SID),

CID INT REFERENCES COURSE(CID),

DOR DATE,

FEE SMALLMONEY,

PRIMARY KEY(SID,CID)

)

INSERT INTO REGISTRATIONS VALUES(1,10,GETDATE(),1000)

INSERT INTO REGISTRATIONS VALUES(1,11,GETDATE(),1000)

INSERT INTO REGISTRATIONS VALUES(2,10,GETDATE(),1000)

INSERT INTO REGISTRATIONS VALUES(1,10,GETDATE(),1000) => ERROR

Exercise :-

PRODUCTS

prodid pname price category brand

CUSTOMERS

custid cname city country

SALES

DATEID PRODID CUSTID QTY AMOUNT

=> identify primary keys & foreign keys and write the create table

commands ?

08-may-21

composite foreign key :-

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

=> if foreign key declared for combination then it is called

composite foreign key.

=> a composite foreign key refers composite primary key.

Example :-

REGISTRATIONS

SID CID DOR FEE

-----------

1 10 ?? 1000

1 11 ?? 1000

2 10 ?? 1000

CERTIFICATES

CERTNO DOI SID CID

1000 ?? 1 10

1001 ?? 1 11

1002 ?? 2 11 => INVALID

=> in the above example SID,CID combination should match with

REGISTRATIONS table SID,CID combination.so declare this

combination as foreign key that refers registrations table

primary key.

CREATE TABLE CERTIFICATES

(

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 declared at table level ?

A UNIQUE

B CHECK

C NOT NULL

D PRIMARY KEY

E FOREIGN KEY

ANS :- C

Adding constraints to existing table :-

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

=> using ALTER command we can add constraints to existing table.

CREATE TABLE emp66

(

empno int,

ename varchar(10),

sal money,

dno int

)

Adding primary key :-

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

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

first change the column to not null

=> add primary key to empno ?

1 ALTER TABLE emp66

ALTER COLUMN empno int NOT NULL

2 ALTER TABLE emp66

ADD PRIMARY KEY(empno)

Adding check constraint :-

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

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

ALTER TABLE emp66

ADD CHECK(sal>=3000)

ALTER TABLE emp

ADD CHECK(sal>=3000) => ERROR

=> while adding constraint sql server also validates existing data.

if existing data satisifies the condition then constraint is

added otherise it returns error.

WITH NOCHECK :-

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

=> if constraint is added WITH NOCHECK then sql server will not

validate existing data , it validates only new data.

ALTER TABLE emp

WITH NOCHECK ADD CHECK(sal>=3000)

Adding foreign key :-

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

=> add foreign key to dno that refers dept55 primary key ?

ALTER TABLE emp66

ADD FOREIGN KEY(dno) REFERENCES dept55(dno)

changing from NULL to NOT NULL :-

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

=> modify column ename to not null ?

ALTER TABLE emp66

ALTER COLUMN ename VARCHAR(10) NOT NULL

Droping constraints :-

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

ALTER TABLE <TABNAME>

DROP CONSTRAINT <NAME>

=> drop check constraint in emp66 table ?

ALTER TABLE emp66

DROP CONSTRAINT CK\_\_emp66\_\_sal\_\_17F790F9

=> drop primary key in dept55 table ?

ALTER TABLE dept55

DROP CONSTRAINT PK\_\_dept55\_\_D876095CEF8F270A => ERROR

DROP TABLE DEPT55 => ERROR

TRUNCATE TABLE DEPT55 => ERROR

=> primary key cannot be dropped if referenced by some fk

=> primary key column cannot be dropped if referenced by some fk

=> primary key table cannot be dropped if referenced by some fk

=> primary key table cannot be truncated if referenced by some fk

10-may-21

DELETE rules :-

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

1 ON DELETE NO ACTION (DEFAULT)

2 ON DELETE CASCADE

3 ON DELETE SET NULL

4 ON DELETE SET DEFAULT

=> above rules are declared with foreign key.

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

is deleted.

ON DELETE NO ACTION :-

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

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

CREATE TABLE dept66

(

dno int primary key,

dname varchar(10)

)

INSERT INTO dept66 VALUES(10,'HR'),(20,'IT')

CREATE TABLE emp66

(

eno int primary key,

dno int references dept66(dno)

)

INSERT INTO emp66 VALUES(1,10),(2,10)

DELETE FROM dept66 WHERE dno=10 => ERROR

scenario :-

-----------

ACCOUNTS

ACCNO NAME BAL

-----

100 A 10000

101 B 20000

LOANS

ID TYPE AMT ACCNO

1 H 30 100

2 C 10 100

Rule :- account cannot be closed if associated with loans.

ON DELETE CASCADE :-

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

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

CREATE TABLE dept66

(

dno int primary key,

dname varchar(10)

)

INSERT INTO dept66 VALUES(10,'HR'),(20,'IT')

CREATE TABLE emp66

(

eno int primary key,

dno int references dept66(dno)

ON DELETE CASCADE

)

INSERT INTO emp66 VALUES(1,10),(2,10)

DELETE FROM dept66 WHERE dno=10 => 1 row affected

SELECT \* FROM emp66 => no rows

scenario :-

-----------

ACCOUNTS

ACCNO NAME BAL

100

101

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

1 W ?? 2000 100

2 D ?? 4000 100

Rule :- when account is closed , along with account delete

transactions.

ON DELETE SET NULL :-

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

=> if parent row is deleted then it is deleted but child rows are

not deleted but fk will be set to null.

CREATE TABLE dept66

(

dno int primary key,

dname varchar(10)

)

INSERT INTO dept66 VALUES(10,'HR'),(20,'IT')

CREATE TABLE emp66

(

eno int primary key,

dno int references dept66(dno)

ON DELETE SET NULL

)

INSERT INTO emp66 VALUES(1,10),(2,10)

DELETE FROM dept66 WHERE dno=10 => 1 row affected

SELECT \* FROM emp66

eno dno

1 null

2 null

scenario :-

PROJECTS

projid name duration cost client

100 ABC 5 YEARS 100 TATA MOTORS

101

102

EMP

empid ename sal projid

1 100

2 101

3 101

rule :- when project is deleted set the employees projid to null.

ON DELETE SET DEFAULT :-

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

=> if parent row is deleted then it is deleted but child rows are

not deleted but fk will be set to default value.

CREATE TABLE dept66

(

dno int primary key,

dname varchar(10)

)

INSERT INTO dept66 VALUES(10,'HR'),(20,'IT')

CREATE TABLE emp66

(

eno int primary key,

dno int DEFAULT 20

references dept66(dno)

ON DELETE SET DEFAULT

)

INSERT INTO emp66 VALUES(1,10),(2,10)

DELETE FROM dept66 WHERE dno=10 => 1 row affected

SELECT \* FROM emp66

eno dno

1 20

2 20

UPDATE rules :-

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

1 ON UPDATE NO ACTION

2 ON UPDATE CASCADE

3 ON UPDATE SET NULL

4 ON UPDATE SET DEFAULT

=> UPDATE rules are also declared with foreign key.

=> these rules specifies how foreign key value is affected if

primary key value is updated.

summary :-

importance of constraints

declaring constraints

column level

table level

adding constraints to existing table

droping constraints

delete rules

update rules

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

11-MAY-21

JOINS

\*\*\*\*\*

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

tables. To get data from two tables we need to join those

two tables.

=> in DB tables are normalized i.e. related data stored in

multiple tables , to gather or to combine data stored in

multiple tables we need to join those tables.

Example :-

ORDERS CUSTOMERS

ordid ord\_dt del\_dt cid cid cname caddr

1000 10- 20- 10 10 a hyd

1001 11- 21- 11 11 b hyd

OUTPUT :-

ordid ord\_dt del\_dt cname caddr

1000 10- 20- a hyd

=> to get the required output we need to join ORDERS & CUSTOMERS

tables.

Types of joins :-

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

1 equi join or inner join

2 outer join

left outer join

right outer join

full outer join

3 non equi join

4 self join

5 cross join or cartisean join

Equi Join or Inner Join :-

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

=> to perform equi join between two tables there must be a common

field and name of the common field need not to be same and pk-fk

relationship is not compulsory.

SELECT columns

FROM tabnames

WHERE join condition

join condition :-

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

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

two tables.

=> join condition decides which record of table1 should be joined

with which record of table2.

WHERE table1.commonfield = table2.commonfield

=> this join is called equi join because here join condition is

based on "=" operator.

Example 1 :-

CREATE TABLE customers

(

cid int primary key,

cname varchar(10),

caddr varchar(10)

)

INSERT INTO customers VALUES(10,'A','HYD'),(11,'B','HYD')

CREATE TABLE orders

(

ordid int PRIMARY KEY,

ord\_dt date,

del\_dt date,

cid int REFERENCES CUSTOMERS(cid)

)

INSERT INTO orders

VALUES(1000,'2021-05-10','2021-05-15',10),

(1001,'2021-05-11;,-2021-05-20',11)

ORDERS CUSTOMERS

ordid ord\_dt del\_dt cid cid cname caddr

1000 10- 20- 10 10 a hyd

1001 11- 21- 11 11 b hyd

write a query to display following output ?

ordid ord\_dt del\_dt cname caddr

1000 10- 20- a hyd

SELECT ordid,ord\_dt,del\_dt,cname,caddr

FROM orders,customers

WHERE orders.cid = customers.cid

display ordid ord\_dt del\_dt cid cname caddr ?

SELECT ordid,ord\_dt,del\_dt,cid,cname,caddr

FROM orders,customers

WHERE orders.cid = customers.cid => ERROR

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 o.ordid,o.ord\_dt,o.del\_dt,

c.cid,c.cname,c.caddr

FROM orders as o,customers as c

WHERE o.cid = c.cid

=> display order details with customer details to be delivered today ?

SELECT o.ordid,o.ord\_dt,o.del\_dt,

c.cid,c.cname,c.caddr

FROM orders as o,customers as c

WHERE o.cid = c.cid /\* join cond \*/

AND

o.del\_dt = GETDATE() /\* filter cond \*/

Example 2 :-

-----------

EMP DEPT

EMPNO ENAME SAL DEPTNO DEPTNO DNAME LOC

7369 smith 920.00 20 10 ACCOUNTS NEW YORK

7499 allen 1920.00 30 20 RESEARCH

7521 ward 1500.00 30 30 SALES

7566 jones 3421.25 20 40 OPERATIONS

7654 martin 1500.00 30

7698 blake 3420.00 30

7782 clark 2695.00 10

=> display EMPNO ENAME SAL DNAME LOC ?

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

EMP DEPT

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

d.dname,d.loc

FROM emp AS e , dept AS d

WHERE e.deptno = d.deptno

=> display employees working at NEW YORK loc ?

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

d.dname,d.loc

FROM emp AS e , dept AS d

WHERE e.deptno = d.deptno

AND

d.loc='NEW YORK'

joining more than 2 tables :-

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

=> when no of tables increses and no of join conditions also increases

to join N tables N-1 join conditions required.

Example :-

EMP DEPT LOCATIONS COUNTRIES

empno deptno locid country\_id

ename dname city country\_name

sal locid state

deptno country\_id

display ename dname city state country ?

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

emp dept locations countries

SELECT e.ename,

d.dname,

l.city,l.state,

c.country\_name

FROM emp e,

dept d,

locations l,

countries c

WHERE e.deptno = d.deptno

AND

d.locid = l.locid

AND

l.country\_id = c.country\_id

=> we can write join queries in 2 styles

1 Native style (SQL SERVER)

2 ANSI style

ANSI style :-

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

=> adv of ANSI style is it gurantees portability.

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

portability.

=> in ANSI style tablenames are seperated by keywords

=> use ON clause for join conditions instead of WHERE clause.

display ENAME DNAME ?

SELECT e.ename,d.dname

FROM emp e INNER JOIN dept d

ON e.depno = d.deptno

display employee names working at NEW YORK loc ?

SELECT e.ename,d.dname

FROM emp e INNER JOIN dept d

ON e.deptno = d.deptno

AND

d.loc='NEW YORK'

SELECT e.ename,d.dname

FROM emp e INNER JOIN dept d

ON e.deptno = d.deptno

WHERE d.loc='NEW YORK'

NOTE :- use ON clause for join conditions

use WHERE clause for filter conditions

outer join :-

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

=> equi join returns only matching records but cannot return unmatched

records , to get unmatched records also perform outer join.

=> outer join is 3 types

1 LEFT OUTER JOIN

2 RIGHT OUTER JOIN

3 FULL OUTER JOIN

EMP DEPT

EMPNO ENAME SAL DEPTNO DEPTNO DNAME LOC

7369 smith 920.00 20 10 ACCOUNTS NEW YORK

7499 allen 1920.00 30 20 RESEARCH

7521 ward 1500.00 30 30 SALES

7566 jones 3421.25 20 40 OPERATIONS => unmatched record

7782 clark 2695.00 10

7902 miller 1300.00 NULL => unmatched record

1 LEFT OUTER JOIN :-

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

=> left outer join returns all rows (matched + unmatched) from

left side table and matching rows from right side table.

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

d.dname,d.loc

FROM emp e LEFT OUTER JOIN dept d

ON e.deptno = d.deptno

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

7369 smith 920.00 RESEARCH ??

7499 allen 1920.00 SALES ??

7521 ward 1500.00 SALES ??

7566 jones 3421.25 RESEARCH ??

7782 clark 2695.00 ACCOUNTS ??

7902 miller 1300.00 NULL NULL => unmatched from emp

2 RIGHT OUTER JOIN :-

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

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

left side table.

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

d.dname,d.loc

FROM emp e RIGHT OUTER JOIN dept d

ON e.deptno = d.deptno

7369 smith 920.00 RESEARCH ??

7499 allen 1920.00 SALES ??

7521 ward 1500.00 SALES ??

7566 jones 3421.25 RESEARCH ??

7782 clark 2695.00 ACCOUNTS ??

NULL NULL NULL OPERATIONS ?? => unmatched from dept

3 full outer join :-

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

=> returns all rows from both tables.

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

d.dname,d.loc

FROM emp e FULL OUTER JOIN dept d

ON e.deptno = d.deptno

7369 smith 920.00 RESEARCH ??

7499 allen 1920.00 SALES ??

7521 ward 1500.00 SALES ??

7566 jones 3421.25 RESEARCH ??

7782 clark 2695.00 ACCOUNTS ??

7902 miller 1300.00 NULL NULL => unmatched from emp

NULL NULL NULL OPERATIONS ?? => unmatched from dept

Assignment :-

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

1 display only unmatched records from left side table ?

2 display only unmatched records from right side table ?

3 display only unmatched records from both tables ?

Non Equi Join :-

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

=> Non Equi Join is performed when tables are not sharing a

common field.

=> this join is called non equi join because here join condition is

not based on "=" operator it is based on > < between operators.

EMP SALGRADE

EMPNO ENAME SAL GRADE LOSAL HISAL

1 A 5000 1 700 1000

2 B 2500 2 1001 2000

3 C 1000 3 2001 3000

4 D 3000 4 3001 4000

5 E 1500 5 4001 9999

display EMPNO ENAME SAL GRADE ?

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

EMP SALGRADE

SELECT empno,ename,sal,grade

FROM emp e JOIN salgrade s

ON e.sal BETWEEN s.losal and s.hisal

1 A 5000 5

2 B 2500 3

3 C 1000 1

4 D 3000 3

5 E 1500 2

14-MAY-21

joining multiple tables in ANSI style :-

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

SELECT columns

FROM t1 join t2

ON condition

join t3

ON condition

join t4

ON condition

display ENAME DNAME GRADE ?

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

EMP DEPT SALGRADE

SELECT e.ename,d.dname,s.grade

FROM emp e INNER JOIN dept d

ON e.deptno = d.deptno

JOIN salgrade s

ON e.sal BETWEEN s.losal and s.hisal ;

SELF JOIN :-

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

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

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

EMP

EMPNO ENAME MGR

7369 smith 7902

7499 allen 7698

7521 ward 7698

7566 jones 7839

7654 martin 7698

7698 blake 7839

7782 clark 7839

7788 scott 7566

7839 king NULL

7902 ford 7566

=> above table contains MGR but to get manager name we need to

perform self join.

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

with different alias.

FROM EMP X,EMP Y

EMP X EMP Y

EMPNO ENAME MGR EMPNO ENAME MGR

7369 smith 7902 7369 smith 7902

7499 allen 7698 7499 allen 7698

7521 ward 7698 7521 ward 7698

7566 jones 7839 7566 jones 7839

7654 martin 7698 7654 martin 7698

7698 blake 7839 7698 blake 7839

7782 clark 7839 7782 clark 7839

7788 scott 7566 7788 scott 7566

7839 king NULL 7839 king NULL

7902 ford 7566 7902 ford 7566

=> display ENAME,MGRNAME ?

SELECT X.ENAME,Y.ENAME AS MGR

FROM EMP X JOIN EMP Y

ON X.MGR = Y.EMPNO

smith ford

allen blake

ward blake

jones king

=> display employees reporting to blake ?

SELECT X.ENAME,Y.ENAME AS MGRNAME

FROM EMP X JOIN EMP Y

ON X.MGR = Y.EMPNO

WHERE Y.ENAME='blake'

=> display blake's manager name ?

SELECT X.ENAME,Y.ENAME AS MGRNAME

FROM EMP X JOIN EMP Y

ON X.MGR = Y.EMPNO

WHERE X.ENAME='blake'

=> display employees earning more than their manager ?

SELECT X.ENAME,Y.ENAME AS MGRNAME

FROM EMP X JOIN EMP Y

ON X.MGR = Y.EMPNO

WHERE X.SAL > Y.SAL

Assignments :-

1 display employees joined before their manager ?

2 display employees earning same salary ?

3 display employees joined on same date ?

4 TEAMS

ID COUNTRY

1 IND

2 AUS

3 RSA

=> write a query to display following output ?

IND VS AUS

IND VS RSA

AUS VS RSA

cross join or cartisean join :-

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

=> cross join returns cross product or cartisean product of two tables

A=1,2

B=3,4

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

=> if cross join performed between two tables then each record of

one table joined with each and every record of another table.

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

SELECT e.ename,d.dname

FROM emp e CROSS JOIN dept d

GROUP BY & JOIN :-

syntax :-

SELECT columns

FROM t1 JOIN t2

ON condition

GROUP BY colname

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

SELECT d.dname,SUM(e.sal)

FROM emp e INNER JOIN dept d

ON e.deptno = d.deptno

GROUP BY d.dname

Assignment :-

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

SALES

DATEID PRODID CUSTID QTY AMOUNT

15-MAY-21 100 10 1 2000

PRODUCTS

PRODID PNAME PRICE CATEGORY

100 ABC 2000 ELECTRONICS

CUSTOMERS

CUSTID NAME ADDR COUNTRY

10 XYZ HYD IND

1 display year wise total sales ?

2 display year wise with in year quarter wise total sales and

display year wise subtotals ?

3 display category wise total sales ?

4 display country wise total sales ?

5 display year wise,country wise,category wise total sales ?

17-may-21

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

=>above operators are performed between records return by two select

statements.

SELECT STATEMENT 1

UNION/UNION ALL/INTERSECT/EXCEPT

SELECT STATEMENT 2

Rules :-

1 no of columns return by both select statements should 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 :-

---------

=> combines rows return by two select statements

=> eliminates duplicates

=> sorts result

SELECT job FROM emp WHERE deptno=20

UNION

SELECT job FROM emp WHERE deptno=30

ANALYST

CLERK

MANAGER

SALESMAN

difference between UNION & JOIN ?

union join

1 horizontal merge vertical merge

2 combines rows combines columns

3 performed between performed between two dissimilar

two similar structures structures

Example :-

T1 T2

F1 F1

1 4

2 5

3 6

T1 U T2 :-

1

2

3

4

5

6

T1 JOIN T2 :-

1 4

2 5

3 6

scenario :-

EMP\_US

eno ename sal dno DEPT

dno dname loc

EMP\_IND

eno ename sal dno

1 display total employees ?

SELECT \* FROM emp\_us

UNION

SELECT \* FROM emp\_ind

2 display employees working at US loc with dept details ?

SELECT e.\*,d.\*

FROM emp\_us e INNER JOIN dept d

ON e.dno = d.dno

3 display total employees with dept details ?

Method 1 :-

SELECT e.\*,d.\*

FROM emp\_ind e INNER JOIN dept d

ON e.dno = d.dno

UNION

SELECT e.\*,d.\*

FROM emp\_us e INNER JOIN dept d

ON e.dno = d.dno

Method 2 :-

SELECT E.\*,D.\*

FROM (SELECT \* FROM emp\_ind

UNION

SELECT \* FROM emp\_us) E INNER JOIN DEPT D

ON E.DNO = D.DNO

UNION ALL :-

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

=> combines rows return by two select statements

=> duplicates are not eliminated

=> result is not sorted

SELECT job FROM emp WHERE deptno=20

UNION ALL

SELECT job FROM emp WHERE deptno=30

diff b/w UNION & UNION ALL ?

UNION UNION ALL

1 eliminates duplicates duplicates are not eliminated

2 sorts result output is not sorted

3 slower faster

INTERSECT :-

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

=> returns common values

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

SELECT job FROM emp WHERE deptno=30

EXCEPT

SELECT job FROM emp WHERE deptno=20

SALESMAN

18-MAY-21

SUBQUERIES / NESTED QUERIES :-

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

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

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

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

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

outer query and result of inner query is input to outer query.

=> use subquery when where condition based on unknown value.

Types of subqueries :-

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

1 single row subqueries

2 multi row subqueries

3 co-related subqueries

4 derived tables

5 scalar subqueries

single row subqueries :-

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

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

subquery.

SELECT columns

FROM tabname

WHERE colname OP (SELECT STATEMENT)

examples :-

1 display employees earning more than blake ?

SELECT \* FROM emp

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

2 display employees who are senior to king ?

SELECT \* FROM emp

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

3 display name of the employee earning max salary ?

SELECT ename

FROM emp

WHERE sal = MAX(sal) => ERROR

NOTE :- aggregate functions are not allowed in where clause and

they are allowed in SELECT,HAVING clauses.

SELECT ename

FROM emp

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

4 display name of the employee having max experience ?

SELECT ename

FROM emp

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

5 display 2nd max salary ?

SELECT MAX(sal)

FROM emp

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

6 display 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))

NOTE :-

=> outer query can be SELECT/INSERT/UPDATE/DELETE but inner query

must be always SELECT.

7 update employee sal to max(sal) of 30th dept whose empno=7499 ?

UPDATE emp

SET sal = (SELECT MAX(sal)

FROM emp

WHERE deptno=30)

WHERE empno=7499

8 swap employee salaries whose empno=7369,7499 ?

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)

20-may-21

Multi-row subqueries :-

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

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

multirow subquery.

SELECT columns

FROM tabname

WHERE colname OP (SELECT STATEMENT)

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

=> display employees whose job = job of smith,blake ?

SELECT \*

FROM emp

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

ANY operator :-

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

=> use ANY operator for comparision with atleast one value.

WHERE X > ANY(1000,2000,3000)

if X=800 FALSE

X=1500 TRUE

X=4500 TRUE

WHERE X < ANY(1000,2000,3000)

IF X=800 TRUE

X=1500 TRUE

X=4500 FALSE

ALL operators :-

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

=> use ALL operator when comparision with ALL values.

WHERE X > ALL(1000,2000,3000)

IF X=800 FALSE

X=1500 FALSE

X=4500 TRUE

WHERE X < ALL(1000,2000,3000)

IF X=800 TRUE

X=1500 FALSE

X=4500 FALSE

=> display employees earning more than all managers ?

SELECT \*

FROM emp

WHERE sal > ALL(SELECT sal FROM emp WHERE job='MANAGER')

=> display employees earning more than atleast one manager ?

SELECT \*

FROM emp

WHERE sal > ANY(SELECT sal FROM emp WHERE job='MANAGER')

SINGLE MULTIPLE

= IN

> >ANY >ALL

< <ANY <ALL

<> NOT IN

co-related subqueries :-

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

=> if inner query refers values of outer query then it is called

co-realated subquery.

=> in co-related subquery execution starts from outer query and

inner query is executed no of times depends on no of rows affected

by outer query.

=> use co-realted subquery to execute the subquery for every row

return by outer query.

1 returns a row from outer query

2 pass value to inner query

3 executes inner query

4 inner query output passed to outer query

5 evaluates outer query where cond

Example :-

EMP

EMPNO ENAME SAL DEPTNO

1 A 5000 10

2 B 3000 20

3 C 4000 30

4 D 6000 20

5 E 3000 10

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

SELECT \*

FROM emp x

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

1 A 5000 10 5000 > (4000) TRUE

2 B 3000 20 3000 > (4500) FALSE

3 C 4000 30 4000 > (4000) FALSE

4 D 6000 20 6000 > (4500) TRUE

5 E 3000 10 3000 > (4000) FALSE

=> display employees who are eaning max(sal) in their dept ?

SELECT \*

FROM emp x

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

21-MAY-21

=> display top 3 max salaries ?

emp a emp b

SAL SAL

5000 5000 3 > (0) TRUE

1000 1000 3 > (4) FALSE

3000 3000 3 > (2) TRUE

2000 2000 3 > (3) FALSE

4000 4000 3 > (1) TRUE

SELECT DISTINCT a.sal

FROM emp a

WHERE 3 > (SELECT COUNT(DISTINCT b.sal)

FROM emp b

WHERE a.sal < b.sal)

ORDER BY sal DESC

=> display 3rd max salary ?

SELECT DISTINCT a.sal

FROM emp a

WHERE (3-1) = (SELECT COUNT(DISTINCT b.sal)

FROM emp b

WHERE a.sal < b.sal)

ORDER BY sal DESC

DERIVED TABLES :-

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

=> subqueries appears in FROM clause are called derived tables.

SELECT columns

FROM (SELECT statement) <alias>

WHERE condition

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

=> derived tables are used in the following scenarios.

1 to control the order of execution of clauses

2 to use the result of one process in another process

3 to join a table with query output

Examples :-

1 display employee names and annual salaries whose annual sal > 30000 ?

SELECT ename,sal\*12 as annsal

FROM emp

WHERE annsal > 30000 => ERROR

=> column alias cannot be referenced by WHERE clause because

WHERE clause is executed before SELECT. To overcome this

problem use derived tables.

SELECT \*

FROM (SELECT ename,sal\*12 as annsal

FROM emp) E

WHERE annsal > 30000

2 display top 5 employees ?

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 \*

FROM (SELECT empno,ename,sal,

DENSE\_RANK() OVER (ORDER BY sal DESC) as rnk

FROM emp) AS E

WHERE rnk<=5

3 display first 5 rows from emp ?

SELECT \*

FROM (SELECT ROW\_NUMBER() over (ORDER BY empno ASC) as rno,

empno,ename,sal

FROM emp) AS E

WHERE rno<=5

display 5th record ?

WHERE rno=5

display 5th to 10th record ?

WHERE rno BETWEEN 5 AND 10

dsiplay even no rows ?

WHERE rno%2=0

display last 3 rows from emp ?

SELECT empno,ename,sal FROM emp

EXCEPT

SELECT empno,ename,sal

FROM (SELECT ROW\_NUMBER() over (ORDER BY empno ASC) as rno,

empno,ename,sal

FROM emp) AS E

WHERE rno <= (SELECT COUNT(\*)-3 FROM emp)

26-MAY-21

delete duplicate rows ?

EMP22

ENO ENAME SAL

1 A 5000

2 B 6000

3 C 7000

1 A 5000 => duplicate row

2 B 6000 => duplicate row

SELECT eno,ename,sal,row\_number() over (partition by eno,ename,sal

order by eno asc) as rno

FROM emp ;

1 A 5000 1

1 A 5000 2

2 B 6000 1

2 B 6000 2

3 C 7000 1

DELETE FROM (SELECT eno,ename,sal,row\_number() over (partition by eno,ename,sal

order by eno asc) as rno

FROM emp) AS E

WHERE RNO > 1 => ERROR

=> derived tables outer query cannot be DML it must be SELECT statement. To overcome

this use CTEs.

CTE :- (common table expression)

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

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

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

=> CTEs are also similar to derived tables to simplify the complex queries.

=> CTEs are used to use the result of one process in another process.

syntax ;-

WITH expression\_name [ ( column\_name [,...n] ) ]

AS

(CTE\_query\_definition )

To view the CTE result we use a Select query with the CTE expression name.

Select [Column1,Column2,Column3 …..] from expression\_name

Example 1 :-

delete first 5 rows from emp table ?

step 1 :- generate row numbers

SELECT empno,ename,sal,

ROW\_NUMBER() OVER (ORDER BY empno ASC) AS RNO

FROM emp ;

7369 1

7499 2

7521 3

7566 4

7654 5

7698 6

7782 7

step 2 :- delete the records whose rno <= 5

WITH emp\_temp

AS

( SELECT empno,ename,sal,

ROW\_NUMBER() OVER (ORDER BY empno ASC) AS RNO

FROM emp )

DELETE FROM emp\_temp WHERE rno<=5

Example 2 :-

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

delete the duplicate records ?

EMP22

ENO ENAME SAL

1 A 5000

2 B 6000

3 C 7000

1 A 5000 => duplicate row

2 B 6000 => duplicate row

STEP 1 :- generate the row numbers

SELECT eno,ename,sal,row\_number() over (partition by eno,ename,sal

order by eno asc) as rno

FROM emp22

1 A 5000 1

1 A 5000 2

2 B 6000 1

2 B 6000 2

3 C 7000 1

STEP 2 :- delete the records whose rno > 1

WITH E

AS

(SELECT eno,ename,sal,row\_number() over (partition by eno,ename,sal

order by eno asc) as rno

FROM emp22)

DELETE FROM E WHERE RNO>1

scalar subqueries :-

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

=> subqueries in SELECT clause are called scalar subqueries.

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

FROM tabname

WHERE cond

=> here the subquery output acts like a column.

=> use scalar subqueries to show the query output in seperate column

Example 1 :-

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

(SELECT COUNT(\*) FROM DEPT) AS DEPT

EMP DEPT

14 4

Example 2 :-

display dept wise total salary ?

SELECT deptno,SUM(sal)

FROM emp

GROUP BY deptno

10 8750

20 10875

30 9400

display deptno,dept\_totsal,totsal ?

SELECT deptno,SUM(sal) as dept\_totsal,(SELECT SUM(sal) FROM emp) as totsal

FROM emp

GROUP BY deptno ;

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 ;

27-may-21

PIVOT operator :-

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

=> used to convert rows into columns.

=> used for cross tabulation or matrix report.

syn :-

SELECT columns

FROM (SELECT required data) AS <ALIAS>

PIVOT

(aggr-expr for col IN (v1,v2,v2,--)) AS <NAME>

ORDER BY col ASC/DESC

Example 1 :- (matrix report)

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

10 20 30

ANALYST NULL 6000 NULL

CLERK 1300 1900 950

MANAGER 2450 2975 2850

SALESMAN NULL NULL 5600

Example 2 :- (matrix report)

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

SELECT \*

FROM (SELECT DATEPART(yy,hiredate) as year,

DATEPART(q,hiredate) as qrt,

empno

FROM emp) AS E

PIVOT

(COUNT(empno) for qrt IN ([1],[2],[3],[4])) AS PIVOT\_TBL

ORDER BY year ASC

1 2 3 4

1980 ? ? ? ?

1981 ? ? ? ?

1982 ? ? ? ?

1983 ? ? ? ?

Example 3 :- converting rows into columns

STUDENT

SNO SNAME SUBJECT MARKS

1 A MAT 80

1 A PHY 70

1 A CHE 90

2 B MAT 90

2 B PHY 60

2 B CHE 70

OUTPUT :-

SNO SNAME MAT PHY CHE

1 A 80 70 90

2 B 90 60 70

SELECT \*

FROM STUDENT

PIVOT

(SUM(MARKS) FOR SUBJECT IN ([MAT],[PHY],[CHE])) AS PIVOT\_TBL

ORDER BY SNO ASC

UNPIVOT :-

----------

=> UNPIVOT is the reverse process of PIVOT

=> used to convert columns into rows.

STUD\_MARKS

SNO SNAME MAT PHY CHE

1 A 80 70 90

2 B 90 60 70

OUTPUT :-

SNO SNAME SUBJECT MARKS

1 A MAT 80

1 A PHY 70

1 A CHE 90

2 B MAT 90

2 B PHY 60

2 B CHE 70

SELECT SNO,SNAME,SUBJECT,MARKS

FROM STUD\_MARKS

UNPIVOT

(MARKS FOR SUBJECT IN ([MAT],[PHY],[CHE])) AS PIVOT\_TBLE

ORDER BY SNO ASC

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

28-may-21 Database Transactions

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

=> a Transaction is a unit of work that contains one or more dmls and that must be

saved as a whole or must be cancelled as a whole.

ex :- money transfer

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

update1(bal=bal-1000) update2(bal=bal+1000)

successful failed INVALID

failed successful INVALID

successful successful VALID

failed failed VALID

=> every db transaction must gurantee a property called atomocity i.e. all or none.

if transaction contains multiple dmls then if all are successful then it must be

saved , if one of the operation fails then entire transaction must be cancelled.

=> the following commands provided by SQL SERVER to control transactions are called

TCL commands (Transaction Control Language).

1 COMMIT => to save transaction

2 ROLLBACK => to cancel transaction

3 SAVE TRANSACTION => to cancel transaction upto some point.

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

=> by default in SQL SERVER a txn begins implicitly with DML command and ends implicity with COMMIT.

=> a transaction can also be explicitly started with "BEGIN TRANSACTION" command

and explicitly can be ended with COMMIT/ROLLBACK command.

BEGIN TRANSACTION

UPDATE1

INSERT1

UPDATE2

INSERT2

COMMIT

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

are saved.

BEGIN TRANSACTION

UPDATE1

INSERT1

UPDATE2

INSERT2

ROLLBACK

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

cancelled.

begin transaction

update1

cnt1 = @@rowcount

update2

cnt2 = @@rowcount

if cnt1=1 and cnt2=1

commit

else

rollback

Questions :-

1

create table a(a int)

begin transaction

insert into a values(10)

insert into a values(20)

insert into a values(30)

insert into a values(40)

rollback

output :- create table => saved

insert => cancelled

2

create table a(a int)

begin transaction

insert into a values(10)

insert into a values(20)

rollback

insert into a values(30)

insert into a values(40)

rollback

output :-

created table => saved

insert 10,20 => cancelled

insert 30,40 => saved

SAVE TRANSACTION :-

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

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

=> using save transaction we can cancel part of the transaction.

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

commit

Analytical functions :-

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

LAG() & LEAD() :-

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

LAG => function returns previous value

LEAD => function returns next value

LAG(EXPR,INT) OVER (ORDER BY COL ASC/DESC)

LEAD(EXPR,INT) OVER (ORDER BY COL ASC/DESC)

Example :-

SELECT empno,ename,sal,

LAG(sal,1) OVER (ORDER BY empno ASC) as prev\_sal

FROM emp ;

7369 SMITH 1300.00 NULL

7499 ALLEN 1100.00 1300.00

7521 WARD 1250.00 1100.00

7566 JONES 2975.00 1250.00

7654 MARTIN 1250.00 2975.00

SELECT empno,ename,sal,

LEAD(sal,1) OVER (ORDER BY empno ASC) as prev\_sal

FROM emp ;

7839 KING 5000.00 1500.00

7844 TURNER 1500.00 1100.00

7876 ADAMS 1100.00 950.00

7900 JAMES 950.00 3000.00

7902 FORD 3000.00 1300.00

7934 MILLER 1300.00 4000.00

9999 ABC 4000.00 NULL

display ENAME,HIREDATE,days ?

SELECT empno,ename,sal,hiredate,

DATEDIFF(DD,LAG(hiredate,1) OVER (ORDER BY hiredate ASC),hiredate) as days

FROM emp

display ename,sal,first value ?

SELECT empno,ename,sal,

FIRST\_VALUE(sal) OVER (ORDER BY empno ASC) AS FIRST

FROM emp

7369 SMITH 1300.00 1300.00

7499 ALLEN 1100.00 1300.00

7521 WARD 1250.00 1300.00

7566 JONES 2975.00 1300.00

7654 MARTIN 1250.00 1300.00

7698 BLAKE 2850.00 1300.00

PERCENT\_RANK() :-

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

=> The PERCENT\_RANK function computes the rank of the employee's salary within a

department as a percentage.

SELECT empno,ename,sal,deptno,

PERCENT\_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS PCT

FROM EMP

DB security :-

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

1 LOGINS => provides security at server level

2 USERS => provides security at db level

3 PRIVILEGES => provides security at table level

4 VIEWS => provides security at row & col level

SERVER (logins)

DATABASE (users)

TABLES (privileges)

DATA (views)

creating logins :-

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

=>

SCHEMA OBJECTS / DATABASE OBJECTS :-

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

1 TABLES

2 VIEWS

3 SYNONYMS

4 SEQUENCES

5 INDEXES

01-JUN-21

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

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

=> views are 2 types

1 simple views

2 complex views

simple views :-

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

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

syn :-

CREATE VIEW <NAME>

AS

SELECT STATEMENT ;

Example 1 :-

CREATE VIEW V1

AS

SELECT empno,ename,job,deptno FROM emp

=> when the above command submitted to sql server , it creates view "v1" and stores

query but not query output

SELECT \* FROM v1

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

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

Granting permissions on view to user :-

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

DBO :-

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

VIJAY :-

1 SELECT \* FROM V1

2 INSERT INTO V1 VALUES(777,'PQR','CLERK',30)

3 UPDATE V1 SET JOB='MANAGER' WHERE EMPNO=777

4 UPDATE V1 SET SAL=2000 WHERE EMPNO=777 => ERROR (can't update sal through view)

Example 2 :- (ROW LEVEL SECURITY)

DBO :-

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','CLERK',30) => 1 row affected

=> above insert command executed successfully even though it's 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.

DBO :-

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(999,'BQR','CLERK',30) => ERROR

complex views :-

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

=> a view said to be complex view

1 if based on multiple tables

2 if query contains group by clause

having clause

distinct clause

aggregate functions

expressions

set operators

subqueries

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

Example 1 :-

CREATE VIEW CV1

AS

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

d.deptno,d.dname,d.loc

FROM emp e INNER JOIN dept d

ON e.deptno = d.deptno

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

writing join query write the simple query as follows.

SELECT \* FROM CV1

02-jun-21

Example 2 :-

CREATE VIEW CV2

AS

SELECT d.dname,MIN(e.sal) as minsal,

MAX(e.sal) as maxsal,

SUM(e.sal) as sumsal,

COUNT(e.empno) as cnt

FROM emp e INNER JOIN dept d

ON e.deptno = d.deptno

GROUP BY d.dname ;

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

SELECT \* FROM CV2

Diff b/w simple and complex views ?

simple complex

1 based on single table based on multiple tables

2 query performs simple query performs complex operations like

operations joins,group by etc

3 updatable i.e. allows DMLs not updatable

display list of views created by user ?

SELECT \* FROM INFORMATION\_SCHEMA.views

Droping view :-

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

DROP VIEW V1

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

ans :- views are not dropped

WITH SCHEMABINDING :-

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

=> if view created with schemabinding then sql server will not allow the user to

drop table if any view exists on the table.

=> follow below rules to create view with schemabinding

1 "\*' is not allowed in SEELCT

2 tablename should be prefixed with schema name.

Example :-

CREATE VIEW V5

WITH SCHEMABINDING

AS

SELECT DEPTNO,DNAME,LOC FROM DBO.DEPT

DROP TABLE DEPT => ERROR

SYNONYMS :-

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

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

=> synonyms are created when tablename is lengthy or complex

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

Ex :- CREATE SYNONYM E FOR EMP

=> after creating synonym instead of using tablename we can use synonym name in

SELECT/INSERT/UPDATE/DELETE queries.

SELECT \* FROM e

Question :-

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

CREATE SYNONYM E FOR EMP

SELECT \* FROM EMP E

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

diff b/w synonym & alias ?

synonym alias

1 permanent not permanent

2 stored in db not stored in db

3 scope of the synonym scope of the alias is upto the query

is upto the schema

display list of synonyms ?

SELECT \* FROM sys.synonyms

03-jun-21

SEQUENCE :-

-----------

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

=> using SEQUENCE we can auto increment column values

syn :-

CREATE SEQUENCE <NAME>

[START WITH <VALUE>]

[INCREMENT BY <VALUE>]

[MAXVALUE <VALUE>]

[MINVALUE <VALUE>]

[CYCLE/NOCYCLE]

ex :- (calling sequence in insert command)

CREATE SEQUENCE S1

START WITH 1

INCREMENT BY 1

MAXVALUE 5

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

CREATE TABLE STUDENT

(

SID INT ,

SNAME VARCHAR(10)

)

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1,'A')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1,'B')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1,'C')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1,'D')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1,'E')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S1,'F') => ERROR

SELECT \* FROM STUDENT

SID SNAME

1 A

2 B

3 C

4 D

5 E

example 2 :- (calling sequence in update command)

CREATE SEQUENCE S2

START WITH 100

INCREMENT BY 1

MAXVALUE 1000

=> use above sequence to make empno sequence

UPDATE emp SET empno = next value for s2

NOCYCLE / CYCLE :-

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

=> by default sequence created with NOCYCLE

=> if sequence created with NOCYCLE then it starts from start with and every time

value is incremented and it generates upto maxvalue and after reaching maxvalue

then it stops.

=> if sequence created with CYCLE then it starts from start with and generates upto

maxvalue and after reaching maxvalue then it will be reset to minvalue.

Ex :-

CREATE SEQUENCE S3

START WITH 1

INCREMENT BY 1

MAXVALUE 5

MINVALUE 1

CYCLE

CREATE TABLE STUDENT

(

SID INT ,

SNAME VARCHAR(10)

)

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'A')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'B')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'C')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'D')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'E')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'F')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'G')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'H')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'I')

INSERT INTO STUDENT VALUES(NEXT VALUE FOR S3,'J')

How to restart sequence ?

CREATE SEQUENCE S5

START WITH 1

INCREMENT BY 1

MAXVALUE 1000

=> after reaching 50 restart the sequence from 1 ?

ALTER SEQUENCE S5 RESTART WITH 1

diff b/w identity and sequence ?

identity sequence

1 not db object db object

2 property is tied to sequence can be shared among multiple tables

a particular table and since it is not tied to a particular column

column and cannot be and table.

shared among multiple

tables

3 to generate next value call NEXT VALUE FOR SEQUENCE to geneate

for identity , a new next value

has to be inserted

into table

4 cannot be called in sequence can be called in update command

update command

5 maxvalue cannot be maxvalue can be defined for sequence

defined for identity

04-jun-21

INDEX :-

---------

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

=> index in db is similar to index in textbook , in textbook using index a

particular topic can located fastly , In db using index a particular

record can be located fastly.

=> sql server uses following methods to access data

1 TABLE SCAN

2 INDEX SCAN

=> in TABLE SCAN sql server scans complete table to find desired record.

In INDEX SCAN sql server scan only half of the table on avg to find

desired record. so INDEX SCAN is much faster than TABLE SCAN.

=> indexes are created on

1 columns that are frequently accessed in where clause

2 columns on which join operation is performed

Types of Indexes :-

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

1 Non Clustered Index

simple

composite

unique

2 Clustered Index

simple non-clustered index :-

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

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

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

Ex :- CREATE INDEX I1 ON EMP(SAL)

=> sql server uses the above index when user submits the following queries

SELECT \* FROM emp WHERE sal=3000 (index scan)

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

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

=> sql server will not use index for the following queries

SELECT \* FROM emp

SELECT \* FROM emp WHERE ename='BLAKE'

SELECT \* FROM emp WHERE sal<>3000

composite index :-

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

=> a index is said to be composiste if it is created on multiple columns.

Ex :- CREATE INDEX I2 ON EMP(DEPTNO,JOB)

EMP

DEPTNO JOB

20 CLERK

30 SALESMAN

30 SALESMAN

20 MANAGER

10 CLERK

20 CLERK

30 CLERK

10 MANAGER

20

10 30

10 CLERK \* 20 CLERK \*,\* 30 CLERK \*

10 MANAGER \* 20 MANAGER \* 30 SALESMAN \*,\*

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

the index i.e. deptno.

SELECT \* FROM emp WHERE deptno=20; (index scan)

SELECT \* FROM emp WHERE deptno=20 and job='CLERK' (index scan)

SELECT \* FROM emp WHERE job='CLERK' (table scan)

Unique Index :-

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

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

created

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

K

G Q

ALLEN \* JAMES \* MARTIN \* SCOTT \*

ADAMS \* KING \* MILLER \* SMITH \*

BLAKE \*

=> SQL SERVER uses above index when where condition based on ename column

SELECT \* FROM emp WHERE ename='BLAKE' ;

=> the following INSERT command fails because it is trying to insert duplicate

INSERT INTO emp(empno,ename,sal) VALUES(555,'BLAKE',4000) => ERROR

what are the different methods to enforce uniqueness ?

1 declare primary key / unique constraint

2 create unique index

=> we don't need to create index on primary key column because primary key

columns are automatically indexed by sql server and sql server creates

a unique index on primary key column and unique indes doesn't allow

duplicates so primary key also doesn't allow duplicates.

07-jun-21

clustered index :-

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

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

index stores actual records.

=> in non clustered index order of the records in table and order of the records in

index is not same but where as in clustered index this order will be same.

Example :-

CREATE TABLE cust

(

cid int,

cname varchar(10)

)

INSERT INTO cust VALUES(10,'A')

INSERT INTO cust VALUES(50,'B')

INSERT INTO cust VALUES(90,'C')

INSERT INTO cust VALUES(30,'D')

INSERT INTO cust VALUES(40,'E')

INSERT INTO cust VALUES(60,'F')

CREATE CLUSTERED INDEX I10 ON CUST(CID)

50

30 70

10 A 40 E 60 F 90 C

30 D 50 B

SELECT \* FROM cust

CID CNAME

10 A

30 D

40 E

50 B

60 F

90 C

=> only one clustered index is allowed per table

=> by default SQL SERVER creates a clustered index on primary key column.

What type of index is created on primary key ?

ans :- unique clustered index

diff b/w non clustered and clustered index ?

non clustered clustered

1 stores pointers to acutal records stores acutal records

2 order of the records in table and order will be same

in index is not same

3 needs extra storage doesn't need extra storage

4 requires two lookup to requires only one lookup

find desired record

5 sql server allows 999 sql server allows only one

non clustered indexes clustered index per table

per table

6 a non clustered unique a clustered unique index is

index is created on created on primary key columns

unique columns

display list of indexes created on table ?

1 sp\_helpindex cust

2 select \* from sys.indexes

Droping index :-

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

drop index emp.i1

if we drop table what about indexes created on table ?

ans :- indexes are also dropped

DB8AM

TABLES

ROWS & COLS

CONSTRAINTS

INDEXES

TRIGGERS

VIEWS

SYNONYMS

SEQUENCES

SCHEMA :-

----------

=> db objects can be logically grouped into one object called schema.

=> default schema in sql server is dbo

CREATE SCHEMA HOME\_LOANS

CREATE TABLE HOME\_LOANS.CUSTOMERS

(

CID INT,

CNAME VARCHAR(10)

)

SERVER

DATABASE

USERS

SCHEMA1

TABLES

VIEWS

SEQUENCES

SCHEMA2

TABLES

VIEWS

SEQUENCES

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

09-jun-21 TSQL programming

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

Features :-

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

1 improves performance :-

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

=> using TSQL programming , SQL commands can be grouped into one block and we can

submit that program to sql server , so no of requests & response between user

and sql server are reduced and performance is improved.

2 supports conditional statements :-

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

=> tsql programming supports conditional statements like IF-THEN-ELSE

3 supports loops :-

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

=> loops are used to executge statements repeatedly multiple times and

tsql programming supports loops like while loop.

4 supports error handling :-

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

=> in tsql programming if any statement causes error then we can handle that

error and we can replace system generated message with our own message.

5 supports reusability :-

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

=> TSQL programs can be stored in db and applications which are connected to db

can reuse the tsql programs.

6 supports security :-

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

=> tsql programs are secured because tsql programs are stored in db , so only

authorized users can execute these programs.

=> TSQL blocks are 2 types

1 anonymous blocks

2 named blocks

procedures

functions

triggers

Anonymous blocks :-

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

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

=> the following statements are used in tsql programs.

1 DECLARE

2 SET

3 PRINT

DECLARE statement :-

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

=> statement is used to declare variables

syn :-

DECLARE @var1 datatype(size)

ex :-

DECLARE @x int

DECLARE @s varchar(10)

DECLARE @d date

DECLARE @x int,@s varchar(10),@d date

10-jun-21

SET statement :-

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

=> statement used to assign value to variable

SET @var = value

SET @x = 100

SET @s = 'abc'

SET @d = GETDATE()

PRINT statement :-

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

=> statement used to print message or values

PRINT 'hello'

PRINT @x

=> 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 = '1947-08-15'

PRINT DATENAME(dw,@d)

DB programming with TSQL :-

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

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

program.

=> the following commands can be executed from tsql program

1 DML (insert,update,delete,merge)

2 DRL (select)

3 TCL (commit,rollback,save transaction)

SELECT command syntax :-

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

SELECT @var1 = col1 ,

@var2 = col2,

-----------

FROM tabname

[WHERE cond]

Ex :-

SELECT @a=ename,

@b=sal

FROM emp

WHERE empno = 7369

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

DECLARE @eno int,@name varchar(10),@sal money

SET @eno=7844

SELECT @name=ename , @sal=sal FROM emp WHERE empno = @eno

PRINT @name + ' ' + CAST(@sal as varchar)

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

DECLARE @eno int,@hire date,@expr tinyint

SET @eno=7844

SELECT @hire=hiredate FROM emp WHERE empno=@eno

SET @expr = DATEDIFF(yy,@hire,GETDATE())

PRINT @expr

conditional statements :-

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

1 if-then-else

2 multi if

3 nested if

1 if-then-else :-

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

if cond

begin

statements

end

else

begin

statements

end

2 mutli-if :-

if cond1

begin

statements

end

else if cond2

begin

statements

end

else if cond3

begin

statements

end

else

begin

statements

end

3 nested if :-

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

if cond

begin

if cond

begin

statements

end

else

begin

statements

end

end

else

begin

statements

end

11-jun-21

write a prog to input empno and increment employee sal by specific amount

and after increment if sal exceeds 5000 then cancel that increment ?

DECLARE @eno int,@amt money,@sal money

SET @eno=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 sno and calculate total,avg and result and insert into

result table ?

STUDENT

SNO SNAME S1 S2 S3

1 A 80 90 70

2 B 30 60 50

RESULT

SNO STOT SAVG SRES

DECLARE @sno int,@s1 int,@s2 int,@s3 int,@tot int,@avg decimal(5,2),@res char(4)

SET @sno=1

SELECT @s1=s1,@s2=s2,@s3=s3 FROM student WHERE sno=@sno

SET @tot = @s1+@s2+@s3

SET @avg = @tot/3,0

IF @s1>=35 AND @s2>=35 AND @s3>=35

SET @res ='PASS'

ELSE

SET @res='FAIL'

INSERT INTO RESULT VALUES(@sno,@tot,@avg,@res)

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

if job=CLERK incr sal by 10%

SALESMAN 15%

MANAGER 20%

OTHERS 5%

DECLARE @eno int,@job VARCHAR(10),@pct int

SET @eno=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

12-JUN-21

ACCOUNTS

ACCNO NAME BAL

100 A 10000

101 B 20000

=> write a prog to process bank transaction (W/D) ?

DECLARE @acno int,@type char(1),@amt money,@bal money

SET @acno=100

SET @type='w'

SET @amt=1000

IF @type='w'

BEGIN

SELECT @bal=bal FROM accounts WHERE accno=@acno

IF @amt > @bal

PRINT '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'

=> write a prog to process money transfer ?

DECLARE @sacno int,@tacno int,@amt money,@bal money

DECLARE @cnt1 int,@cnt2 int

SET @sacno=100

SET @tacno=101

SET @amt=1000

SELECT @bal=bal FROM accounts WHERE accno=@sacno

IF @amt > @bal

PRINT 'insufficient balance'

ELSE

BEGIN

BEGIN TRANSACTION

UPDATE accounts SET bal=bal-@amt WHERE accno=@sacno

SET @cnt1 = @@ROWCOUNT

UPDATE accounts SET bal=bal+@amt WHERE accno=@tacno

SET @cnt2 = @@ROWCOUNT

IF @cnt1=1 and @cnt2=1

COMMIT

ELSE

ROLLBACK

END

what is @@rowcount ?

=> it is a system variable that returns no of rows affected by last dml command.

=> all system variables are prefixed with "@@"

WHILE loop :-

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

=> loops are used to execute statements repeatedly multiple times.

=> TSQL supports only one loop i.e. WHILE loop.

WHILE(cond)

BEGIN

statements

END

=> if cond = true loop continues

if cond = false loop terminates

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

DECLARE @x int = 1

WHILE(@x<=20)

BEGIN

PRINT @x

SET @x = @x+1

END

15-jun-21

=> write a prog to print 2021 calendar ?

date day

2021-01-01 ?

2021-01-02 ?

2021-12-31 ?

DECLARE @d1 date,@d2 date

SET @d1 = '2021-01-01'

SET @d2 = '2021-12-31'

WHILE(@d1<=@d2)

BEGIN

PRINT CAST(@d1 as VARCHAR) + ' ' + DATENAME(dw,@d1)

SET @d1 = DATEADD(DD,1,@d1)

END

=> write a prog to print only sundays ?

DECLARE @d1 date,@d2 date,@cnt int=0

SET @d1 = '2021-01-01'

SET @d2 = '2021-12-31'

WHILE(@d1<=@d2)

BEGIN

IF DATENAME(dw,@d1)='sunday'

BEGIN

PRINT CAST(@d1 as VARCHAR) + ' ' + DATENAME(dw,@d1)

SET @cnt = @cnt + 1

END

SET @d1 = DATEADD(DD,1,@d1)

END

PRINT 'Total No of Sundays = ' + CAST(@cnt as varchar)

=> write a prog to print following pattern ?

input :- NARESH

output :- N

A

R

E

S

H

DECLARE @s1 varchar(20),@x int = 1

SET @s1 = 'NARESH'

WHILE(@x<=LEN(@s1))

BEGIN

PRINT SUBSTRING(@s1,@x,1)

SET @x = @x + 1

END

=> write a prog to print following pattern ?

INPUT :- NARESH

OUTPUT :- N

NA

NAR

NARE

NARES

NARESH

DECLARE @s1 varchar(20),@x int = 1

SET @s1 = 'NARESH'

WHILE(@x<=LEN(@s1))

BEGIN

PRINT SUBSTRING(@s1,1,@x)

SET @x = @x + 1

END

=> write a prog to print following pattern ?

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

16-jun-21

CURSORS :-

----------

=> CURSORS are used to process row-by-row in TSQL program.

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

data from table and loads that data into temporary memory area and using cursor

we can give name to memory and access row-by-row into tsql program and process the row.

=> follow below steps to use cursor

1 declare cursor

2 open cursor

3 fetch records

4 close cursor

5 deallocate cursor

Declaring cursor :-

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

syn :- DECLARE C1 CURSOR FOR SELECT STATEMENT

ex :- DECLARE C1 CURSOR FOR SELECT ename,sal FROM EMP

Opening cursor :-

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

open <cursor-name>

open c1

1 select stmt associated with cursor is submitted to sql server

2 sql server executes the query and data returned by query is copied to temporary memory

3 cursor c1 points to that memory

Fetching records from cursor :-

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

=> "fetch" statement is used to fetch record from cursor.

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

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

=> a fetch statement fetches one row at a time but to process multiple rows fetch

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

closing cursor :-

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

close <cursor-name>

close c1

Deallocating cursor :-

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

deallocate <cursor-name>

deallocate c1

@@FETCH\_STATUS :-

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

=> it is a system variable that returns

0 => if fetch successful

-1 => if fetch unsuccessful

Examples :-

=> write a prog to print all employee names and salaries ?

DECLARE C1 CURSOR FOR SELECT ename,sal FROM emp

DECLARE @name varchar(20),@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

=> program to insert cursor data into another temporary table ?

DECLARE C1 CURSOR FOR SELECT ename,sal FROM emp

DECLARE @name varchar(20),@sal money

create table #temp

(

ename varchar(10),

sal money

)

OPEN C1

FETCH NEXT FROM C1 INTO @name,@sal

WHILE(@@FETCH\_STATUS=0)

BEGIN

INSERT INTO #temp VALUES(@name,@sal)

FETCH NEXT FROM C1 INTO @name,@sal

END

SELECT \* FROM #temp

CLOSE C1

DEALLOCATE C1

17-JUN-21

=> write a prog to calculate total sal without using SUM function ?

DECLARE C1 CURSOR FOR SELECT SAL FROM EMP

DECLARE @SAL MONEY,@TOTSAL MONEY=0

OPEN C1

FETCH NEXT FROM C1 INTO @SAL

WHILE(@@FETCH\_STATUS=0)

BEGIN

SET @TOTSAL = @TOTSAL + @SAL

FETCH NEXT FROM C1 INTO @SAL

END

PRINT @TOTSAL

CLOSE C1

DEALLOCATE C1

=> write a prog to print employee names as follows ?

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

DECLARE C1 CURSOR FOR SELECT ename FROM emp

DECLARE @name varchar(10),@s varchar(1000)

OPEN C1

FETCH NEXT FROM C1 INTO @name

WHILE(@@FETCH\_STATUS=0)

BEGIN

SET @s = @s + @name + ','

FETCH NEXT FROM C1 INTO @name

END

CLOSE C1

DEALLOCATE C1

PRINT @s

STRING\_AGG() :-

=> function provided by sql server used to concatenate column values.

SELECT STRING\_AGG(ename,',') FROM emp

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

=> Display dept wise employee names ?

SELECT deptno,STRING\_AGG(ename,',') as names

FROM emp

GROUP BY deptno

=> write a prog to increment employee salaries based on the pct in raise\_salary

table ?

RAISE\_SALARY

EMPNO PCT

7369 10

7499 20

7521 12

7566 15

7654 20

DECLARE C1 CURSOR FOR SELECT empno,pct FROM raise\_salary

DECLARE @eno int,@pct int

OPEN C1

FETCH NEXT FROM C1 INTO @eno,@pct

WHILE(@@FETCH\_STATUS=0)

BEGIN

UPDATE emp SET sal = sal + (sal\*@pct/100) WHERE empno=@eno

FETCH NEXT FROM C1 INTO @eno,@pct

END

CLOSE C1

DEALLOCATE C1

Assignments :-

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

result table ?

STUDENT

SNO SNAME S1 S2 S3

1 A 80 90 70

2 B 30 60 50

RESULT

SNO TOTAL AVG RESULT

=> write a prog to calculate max salary without using max function ?

=> write a prog to calculate min salary without using min function ?

=> write a prog to calculate total amount of particular order ?

INPUT :- ORDID = 10000

OUTPUT :- AMOUNT = 8000

PRODUCTS

PRODID PNAME PRICE

100 A 1000

101 B 1500

102 C 2000

ORDERS

ORDID PRODID QTY

1000 100 3

1000 101 2

1000 102 1

1001 100 1

18-jun-21

SCROLLABLE CURSOR :-

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

=> by default cursor is called forward only cursor and it supports forward navigation

but doesn't support backward navigation.

=> if cursor declared with SCROLL then it is called scrollable cursor and it supports

both forward and backward navigation.

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

the following fetch statements.

FETCH FIRST => fetches first record

FETCH NEXT => fetches next record

FETCH PRIOR => fetches previous record

FETCH LAST => fetches last record

FETCH ABSOLUTE N => fetches Nth record from starting from 1st record

FETCH RELATIVE N => fetches Nth record from starting from current record

Example 1 :

DECLARE C1 CURSOR SCROLL FOR SELECT ename FROM emp

DECLARE @name VARCHAR(10)

OPEN C1

FETCH FIRST FROM C1 INTO @name

PRINT @name

FETCH ABSOLUTE 5 FROM C1 INTO @name

PRINT @name

FETCH RELATIVE 5 FROM C1 INTO @name

PRINT @name

FETCH LAST FROM C1 INTO @name

PRINT @name

FETCH PRIOR FROM C1 INTO @name

PRINT @name

CLOSE C1

DEALLOCATE C1

Example 2 :-

=> write a prog to print names in reverse order (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

Example 3 :-

write a prog to display every 5th record ?

DECLARE C1 CURSOR SCROLL FOR SELECT ename FROM emp

DECLARE @name VARCHAR(10)

OPEN C1

FETCH RELATIVE 5 FROM C1 INTO @name

WHILE(@@FETCH\_STATUS=0)

BEGIN

PRINT @name

FETCH RELATIVE 5 FROM C1 INTO @name

END

CLOSE C1

DEALLOCATE C1

STATIC VS DYNAMIC :-

static :-

---------

=> if the cursor is static then we if we make changes to base table the changes

are not reflected to cursor.

7844 turner 1500

DECLARE C1 CURSOR STATIC FOR SELECT SAL FROM EMP WHERE EMPNO=7844

DECLARE @SAL MONEY

OPEN C1

UPDATE EMP SET SAL=2000 WHERE EMPNO=7844

FETCH C1 INTO @SAL

PRINT @SAL

CLOSE C1

DEALLOCATE C1

output :- 1500

Dynamic :-

-----------

=> if cursor is dynamic then if we make changes to base table the changes are

reflected to cursor.

=> by default cursor is dynamic

7844 turner 1500

DECLARE C1 CURSOR DYNAMIC FOR SELECT SAL FROM EMP WHERE EMPNO=7844

DECLARE @SAL MONEY

OPEN C1

UPDATE EMP SET SAL=2000 WHERE EMPNO=7844

FETCH C1 INTO @SAL

PRINT @SAL

CLOSE C1

DEALLOCATE C1

OUTPUT :- 2000

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

Exception Handling / Error Handling :-

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

1 syntax errors

2 logical errors

3 runtime errors (exceptions)

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

ex :- declare @x tinyint

set @x=1000 => runtime error

=> in TSQL program if any statement causes runtime error then sql server displays

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

user friendly message then we need to handle that exception.

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

BEGIN TRY

statements => causes exception

END TRY

BEGIN CATCH

statements => handles exception

END CATCH

Example 1 :-

DECLARE @a tinyint,@b tinyint,@c tinyint

BEGIN TRY

SET @a=10

SET @b=0

SET @c = @a/@b

PRINT @C

END TRY

BEGIN CATCH

PRINT 'ERROR'

END CATCH

output :- ERROR

DECLARE @a tinyint,@b tinyint,@c tinyint

BEGIN TRY

SET @a=1000

SET @b=0

SET @c = @a/@b

PRINT @C

END TRY

BEGIN CATCH

PRINT 'ERROR'

END CATCH

output :- ERROR

19-jun-21

Error Handling Functions :-

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

1 ERROR\_NUMBER() => returns error number

2 ERROR\_MESSAGE() => returns error message

3 ERROR\_SEVERITY() => returns error severity level

4 ERROR\_STATE() => returns error state

5 ERROR\_LINE() => returns error line number

Example 1 :-

DECLARE @a tinyint,@b tinyint,@c tinyint

BEGIN TRY

SET @a=1000

SET @b=0

SET @c = @a/@b

PRINT @C

END TRY

BEGIN CATCH

IF ERROR\_NUMBER()=220

PRINT 'value exceeding limit'

ELSE IF ERROR\_NUMBER()=8134

PRINT 'divisor cannot be zero'

END CATCH

list of errors ?

SELECT \* FROM SYS.MESSAGES

Example 2 :-

CREATE TABLE emp66

(

eno int primary key,

ename varchar(10) not null,

sal money check(sal>=3000)

)

write a prog to insert data into emp66 table ?

DECLARE @eno int,@name varchar(10),@sal money

BEGIN TRY

SET @eno=100

SET @name='A'

SET @sal=4000

INSERT INTO emp66 VALUES(@eno,@name,@sal)

END TRY

BEGIN CATCH

IF ERROR\_NUMBER()=2627

PRINT 'eno should not be duplicate'

ELSE IF ERROR\_NUMBER()=515

PRINT 'name should not be null'

ELSE IF ERROR\_NUMBER()=547

PRINT 'sal must be min 3000'

END CATCH

USER DEFINED ERRORS :-

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

=> errors raised by user are called user defined errors

=> when predefined errors not meeting our requirements then we define our own errors called user defined errors

=> user defined errors can be raise by using RAISERROR procedure

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

severity level => 1 to 25

state => 1 to 255

21-jun-21

Example :-

DECLARE @eno int,@amt money

SET @eno=7844

SET @amt=1000

IF DATENAME(dw,GETDATE()) IN ('SATURDAY','SUNDAY')

RAISERROR('weekends not allowed',15,1)

ELSE

UPDATE emp SET sal=sal+@amt WHERE empno=@eno

how to add user defined error to sys.messages table ?

sp\_addmessage 50001,15,'weekends not allowed'

DECLARE @eno int,@amt money

SET @eno=7844

SET @amt=1000

IF DATENAME(dw,GETDATE()) IN ('SATURDAY','SUNDAY')

RAISERROR(50001,15,1)

ELSE

UPDATE emp SET sal=sal+@amt WHERE empno=@eno

how to remove user defined errors from sys.messages table ?

sp\_dropmessage 50001

Assignment :-

=> write a prog to transfer amount from one account to another account ?

ACCOUNTS

ACCNO NAME BAL

100 A 10000

101 B 20000

validations :-

1 check for the existence of source account

2 check for the existence of target account

3 check source balance is sufficient or not

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

22-jun-21

Named TSQL blocks :-

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

1 stored procedures

2 stored functions

3 triggers

sub-programs :-

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

stored procedures

stored functions

Advantages :-

1 modular programming :-

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

=> using procedures/functions a big tsql program can be divided into small modules.

2 reusability :-

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

=> proc/func can be stored in db , so applications which are connected to db can

reuse these proc/func.

3 security :-

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

=> proc/func are secured because they are store in db. so only authorized users can

execute these programs.

4 invoked from front-end applications :-

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

=> proc/func can be called from front-end applications like java,.net

5 improves performance :-

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

=> proc improves performance because they are precompiled i.e. compiled already and

ready for execution. when we create a procedure program is compiled and stored in

db and whenever we call procedure only execution is repeated but not compilation.

so this improves performance.

stored procedures :-

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

=> a procedure is a named TSQL block that accepts some input performs some action

on database and may or may not returns a value.

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

=> db transactions are implemented by using stored procedures.

syntax :-

CREATE OR ALTER PROCEDURE <NAME>

parameters if any

AS

STATEMENTS

parameters :-

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

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

=> parameters are 2 types

1 INPUT

2 OUTPUT

INPUT :-

---------

=> input parameter always recieves value

=> default

=> read only

OUTPUT :-

---------

=> output parameter always sends value

=> write only

Example 1 :-

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

CREATE OR ALTER PROCEDURE update\_salary

@eno int,

@amt money

AS

UPDATE emp SET sal = sal+@amt WHERE empno=@eno

procedure created (compiled + stored in db)

Execution :-

EXECUTE procname parameters

EXECUTE update\_salary 7369,1000

23-JUN-21

Example 2 :- (output parameter)

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

=> create a procedure to increment specific employee sal by specific amount and

after increment send the updated sal to calling program ?

CREATE OR ALTER PROCEDURE update\_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 :-

positional notation :-

DECLARE @s money

EXECUTE update\_salary 7369,1000,@s OUTPUT

PRINT @s

named notation :-

DECLARE @s money

EXECUTE update\_salary @eno=7369,@amt=1000,@newsal=@s OUTPUT

PRINT @s

=> named notation is convienient than positional notation because in named

notation we can pass values to parameters in any order.

Example 3 :-

CREATE TABLE emp77

(

empno int primary key,

ename varchar(10) not null,

sal money check(sal>=3000)

)

=> create a procedure to insert record into emp77 table ?

CREATE OR ALTER PROCEDURE insert\_emp77

@eno int,

@name varchar(10),

@sal money,

@msg varchar(100) OUTPUT

AS

BEGIN TRY

INSERT INTO emp77 VALUES(@eno,@name,@sal)

SET @msg = 'record inserted successfully'

END TRY

BEGIN CATCH

SET @msg=ERROR\_MESSAGE()

END CATCH

Example 4 :-

ACCOUNTS

ACCNO BAL

100 10000

101 20000

=> create a procedure for money withdrawl ?

CREATE OR ALTER PROCEDURE debit

@acno int,

@amt money,

@newbal money OUTPUT

AS

DECLARE @bal money

IF NOT EXISTS (SELECT \* FROM accounts WHERE accno=@acno)

RAISERROR('account does not exists',15,1)

ELSE

BEGIN

SELECT @bal=bal FROM accounts WHERE accno=@acno

IF @amt > @bal

RAISERROR('insufficient balance',15,1)

ELSE

BEGIN

UPDATE accounts SET bal=bal-@amt WHERE accno=@acno

SELECT @newbal=bal FROM accounts WHERE accno=@acno

END

END

Assignment :-

create procedure for money deposit ?

create procedure for money transfer ?

24-JUN-21

Declaring parameters with default value :-

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

=> if parameter declared with default value , while calling procedure if we don't

pass value to parameter then sql server assigns default value.

=> only INPUT parameters can be declared with default value.

Example :-

CREATE OR ALTER PROCEDURE update\_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 :-

postional notation :-

DECLARE @s money

EXECUTE update\_salary 7369,default,@s OUTPUT

PRINT @s

named notation :-

DECLARE @s money

EXECUTE update\_salary @eno=7369,@newsal=@s OUTPUT

PRINT @s

USER DEFINED FUNCTION :-

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

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

functions not meeting our requirements then we create our own functions called

user define functions.

=> a function is also a named TSQL block that accepts some input performs some

calculation and must return a value.

=> functions are created

1 to compute a value

2 to fetch value from db

=> functions are 2 types

1 scalar valued functions(SVF)

2 table valued functions (TVF)

scalar valued functions :-

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

=> if function returns one value then it is called scalar valued function

Syntax :-

CREATE OR ALTER FUNCTION <NAME>(parameters) RETURNS <type>

AS

BEGIN

statements

RETURN <expr>

END

Example 1 :-

CREATE OR ALTER

FUNCTION CALC(@a int,@b int,@op char(1)) RETURNS int

AS

BEGIN

DECLARE @c int

IF @op='+'

SET @c=@a+@b

ELSE IF @op='-'

SET @c=@a-@b

ELSE IF @op='\*'

SET @c=@a\*@b

ELSE

SET @c=@a/@b

RETURN @c

END

Execution :-

1 sql commands

2 another tsql block

3 front-end applications

executing from sql commands :-

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

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

Example 2 :-

create a function to test whether given year is leap year or not ?

CREATE OR ALTER FUNCTION IS\_LEAP(@y int) RETURNS VARCHAR(20)

AS

BEGIN

DECLARE @s varchar(20)

IF @y%4=0 AND @y%400=0 AND @y%100 <> 0

SET @s = 'leap year'

ELSE

SET @s = 'Not a Leap Year'

RETURN @s

END

execution :-

SELECT DBO.IS\_LEAP(2020) => leap year

-

Example 3 :-

ACCOUNTS

ACCNO BAL

100 10000

101 20000

create a function that accepts accno and returns bal ?

CREATE OR ALTER FUNCTION getBalance(@acno int) RETURNS money

AS

BEGIN

DECLARE @bal MONEY

SELECT @bal=bal FROM accounts WHERE accno=@acno

RETURN @bal

END

create a function to test whether account exists or not ?

CREATE OR ALTER FUNCTION getAcct(@acno int) RETURNS tinyint

AS

BEGIN

DECLARE @x tinyint

IF EXISTS (SELECT \* FROM accounts WHERE accno=@acno)

SET @x=1

ELSE

SET @x=0

RETURN

END

25-JUN-21

create a procedure for money withdrawl ?

CREATE OR ALTER PROCEDURE debit

@acno int,

@amt money,

@newbal money OUTPUT

AS

IF dbo.getAcct(@acno)=0

RAISERROR('account does not exists',15,1)

ELSE

BEGIN

IF @amt > dbo.getBalance(@acno)

RAISERROR('insufficient balance',15,1)

ELSE

BEGIN

UPDATE accounts SET bal=bal-@amt WHERE accno=@acno

SET @newbal = dbo.getBalance(@acno)

END

END

Assignment :-

1 create function that accepts name and returns first name ?

input :- sachin tendulkar

output :- sachin

2 create function that accepts name and returns last name ?

input :- sachin tendulkar

output :- tendulkar

3 create a procedure to copy data from custs to custt ?

CUSTS

CID NAME

10 sachin tendulkar

11 virat kohli

CUSTT

CID FNAME LNAME

4

products

prodid pname price

100 A 1500

101 B 2000

102 C 1000

orders

ordid prodid qty

1000 100 3

1000 101 2

1000 102 1

1001 100 2

create function that accept ordid and returns order amount ?

input :- ordid = 1000

output :- amount = 9500

TABLE VALUED FUNCTIONS :-

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

=> table valued functions returns multiple rows

=> return type of these function must be table.

=> return expression must be select statement.

=> table valued functions allows only one statement and that statement must return statement

=> table valued functions are invoked in FROM clause.

syntax :-

CREATE OR ALTER FUNCTION <NAME>(parameters if any) RETURNS TABLE

AS

RETURN (SELECT STATEMENT)

Example 1 :-

create a function that accepts deptno and returns list of employees working for that

dept ?

CREATE OR ALTER FUNCTION getEmpList(@d int) RETURNS TABLE

AS

RETURN (SELECT \* FROM emp WHERE deptno=@d)

execution :-

SELECT \* FROM DBO.getEmpList(20)

Example 2 :-

create a function that returns Top N employee list ?

CREATE OR ALTER FUNCTION getTopNEmpList(@n int) RETURNS TABLE

AS

RETURN ( SELECT \*

FROM (SELECT empno,ename,sal,

DENSE\_RANK() OVER (ORDER BY sal DESC) AS rnk

FROM emp) E

WHERE rnk<=@n)

28-jun-21

Example 3 :-

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

1 W 28- 2000 100

=> create function that returns latest N transactions of particular customer ?

CREATE OR ALTER FUNCTION getLatestNTrans(@acno int,@n int) RETURNS TABLE

AS

RETURN (SELECT \*

FROM (SELECT trid,ttype,tdate,tamt,row\_number() over (order by tdate desc) as rno

FROM transactions

WHERE accno = @acno)

WHERE rno <= @n)

difference between procedures & functions ?

procedure function

1 may or may not returns must return a value

a value

2 can return multiple value always returns one value

3 returns values using returns value using return statement

OUTPUT parameter

4 cannot be called from can be called from sql commands

sql commands

5 allows dmls doesn't allow dmls

6 created to performs dmls created to compute value

7 create procedure to update create function to get balance

balance

difference between scalar valued and table valued function ?

SCALAR TABLE

1 returns one value returns records

2 return type must scalar return type is always TABLE

types like int,varchar

etc

3 return expression is return expression is select statement

scalar variable

4 invoked in SELECT clause invoked in FROM clause

Assignment :-

CUSTOMERS

CUSTID NAME ADDR PHONE EMAILID AADHARNO PANNO

ACCOUNTS

ACCNO ACTYPE BAL CUSTID

TRANSACTIONS

TRID TTYPE TDATE TAMT ACCNO

=> create following proc & func to implement varioub bank transactions

1 account opening (procedure)

2 account closing (procedure)

3 balance enquiry (scalar valued function)

4 money deposit (procedure)

5 money withdrawl (procedure)

6 money transfer (procedure)

7 statement between two given dates (table valued function)

7 latest N transactions (table valued functions)

droping procedure :-

DROP PROCEDURE debit

droping function :-

DROP FUNCTION dbo.CALC

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

29-JUN-21

TRIGGERS :-

-----------

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

by SQL SERVER whenever user submits DML/DDL commands.

=> triggers are created

1 triggers are created to control dmls

2 triggers are created to enforce complex rules and validations

3 triggers are created to audit tables

4 triggers are created to manage replicas

5 triggers are created to generate values for primary key columns

syntax :-

CREATE OR ALTER TRIGGER <NAME>

ON <TABNAME>

AFTER / INSTEAD OF INSERT,UPDATE,DELETE

AS

STATEMENTS

AFTER triggers :-

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

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

INSTEAD OF triggers :-

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

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

Example 1 :-

create trigger to not to allow dmls on emp table on sudnay ?

CREATE OR ALTER TRIGGER T1

ON EMP

AFTER INSERT,UPDATE,DELETE

AS

IF DATENAME(dw,GETDATE())='SUNDAY'

BEGIN

ROLLBACK

RAISERROR('sunday not allowed',15,1)

END

Testing :-

UPDATE emp SET sal=3000 WHERE empno=7369 => ERROR

Example 2 :-

create trigger to not to allow dmls on emp table as follows ?

MON - FRI <10AM AND >4PM

SAT <10AM AND >2PM

SUN -------------

CREATE OR ALTER TRIGGER T2

ON EMP

AFTER INSERT,UPDATE,DELETE

AS

IF DATEPART(DW,GETDATE()) BETWEEN 2 AND 6

AND

DATEPART(HH,GETDATE()) NOT BETWEEN 10 AND 16

BEGIN

ROLLBACK

RAISERROR('only between 10am and 4pm',15,1)

END

ELSE IF DATEPART(DW,GETDATE())=7

AND

DATEPART(HH,GETDATE()) NOT BETWEEN 10 AND 14

BEGIN

ROLLBACK

RAISERROR('only between 10am and 2pm',15,1)

END

ELSE IF DATEPART(DW,GETDATE())=1

BEGIN

ROLLBACK

RAISERROR('sunday not allowed',15,1)

END

Example 3 :-

create trigger to not to allow to update empno ?

CREATE OR ALTER TRIGGER T3

ON EMP

AFTER UPDATE

AS

IF UPDATE(empno)

BEGIN

ROLLBACK

RAISERROR('empno cannot be updated',15,1)

END

Testing :-

UPDATE emp SET empno=9999 WHERE empno=7369 => ERROR

30-JUN-21

Magic tables :-

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

1 INSERTED

2 DELETED

=> with the help of magic tables in triggers we can access data affected by dmls.

=> the record user is trying to insert is copied to inserted table.

INSERT INTO emp VALUES(100,'ABC','CLERK',4000,---) => INSERTED

empno ename job sal

100 ABC CLERK 4000

=> the record user is trying to delete is copied to deleted table.

DELETE FROM emp WHERE empno=7369; => DELETED

empno ename job sal

100 ABC CLERK 4000

=> the record user is trying to update is copied to both INSERTED & DELETED tables

7369 800

UPDATE emp SET sal=2000 WHERE empno=7369; => DELETED

empno sal

7369 800

INSERTED

empno sal

7369 2000

Example 4 :-

=> create trigger to not to allow user to decrement salary ?

CREATE OR ALTER TRIGGER T4

ON EMP

AFTER UPDATE

AS

DECLARE @newsal MONEY,@oldsal MONEY

SELECT @oldsal=sal FROM DELETED

SELECT @newsal=sal FROM INSERTED

IF @newsal < @oldsal

BEGIN

ROLLBACK

RAISERROR('sal cannot be decremented',15,1)

END

Testing :-

UPDATE emp SET sal=1000 WHERE empno=7499 => ERROR

Example 5 :-

CREATE TABLE emp\_resign

(

empno int,

ename varchar(10),

hiredate date,

dor date

)

EMP\_RESIGN

empno ename hiredate dor

create trigger to insert details into emp\_resign table when employee resigns ?

CREATE TRIGGER T5

ON EMP

AFTER DELETE

AS

BEGIN

INSERT INTO emp\_resign

SELECT empno,ename,hiredate,getdate() FROM DELETED

END

Testing :-

DELETE FROM emp where empno=7369

select \* from emp\_resign

7369 SMITH 1980-12-17 2021-06-30

Example 6 :-

EMP66

EMPID ENAME SAL COMM HIREDATE

CREATE TABLE emp66

(

empid int,

ename varchar(10),

sal money,

comm money,

hiredate date

)

CREATE TRIGGER T6

ON EMP66

INSTEAD OF INSERT

AS

DECLARE @name varchar(10),@sal money

DECLARE @eid int,@comm money,@d date

SELECT @name=ename,@sal=sal FROM INSERTED

SELECT @eid = ISNULL(MAX(empid),0) + 1 FROM emp66

SET @comm = @sal\*0.1

SET @d = GETDATE()

INSERT INTO emp66 VALUES(@eid,@name,@sal,@comm@d)

Testing :-

INSERT INTO emp66(ename,sal) VALUES('A',5000) ======> INSERTED table

SELECT \* FROM emp66

empid ename sal comm hiredate

1 A 5000.00 500.00 2021-06-30

01-jul-21

Auditing :-

=> monitoring day-to-day activities on tables

=> day-to-day activities are captured in tables called audit tables

EMP\_AUDIT

UNAME OPERATION OPTIME NEW\_ENO NEW\_ENAME NEW\_SAL OLD\_ENO OLD\_ENAME OLD\_SAL

CREATE TABLE EMP\_AUDIT

(

UNAME VARCHAR(10),

OPERATION VARCHAR(10),

OPTIME DATETIME,

NEW\_ENO INT,

NEW\_ENAME VARCHAR(10),

NEW\_SAL MONEY,

OLD\_ENO INT,

OLD\_ENAME VARCHAR(10),

OLD\_SAL MONEY

)

create trigger to insert details into emp\_audit whenever user insert,update,delete

on emp ?

CREATE OR ALTER TRIGGER T7

ON EMP

AFTER INSERT,UPDATE,DELETE

AS

DECLARE @neweno int,@newename varchar(10),@newsal money

DECLARE @oldeno int,@oldename varchar(10),@oldsal money

DECLARE @op varchar(10)

DECLARE @cnt1 int,@cnt2 int

SELECT @neweno=empno,@newename=ename,@newsal=sal FROM INSERTED

SELECT @oldeno=empno,@oldename=ename,@oldsal=sal FROM DELETED

SELECT @cnt1=COUNT(\*) FROM INSERTED

SELECT @cnt2=COUNT(\*) FROM DELETED

IF @cnt1=1 AND @cnt2=0

SET @op='INSERT'

ELSE IF @cnt1=0 AND @cnt2=1

SET @op='DELETE'

ELSE

SET @op='UPDATE'

INSERT INTO emp\_audit VALUES(user\_name(),@op,GETDATE(),@neweno,@newename,@newsal,

@oldeno,@oldename,@oldsal)

Testing :-

insert into emp(empno,ename,sal) values(9999,'pqr',4000)

update emp set sal=5000 where empno=9999

delete from emp where empno=9999

select \* from emp\_audit

dbo INSERT 2021-06-28 08:29:15.547 9999 pqr 4000.00 NULL NULL NULL

dbo UPDATE 2021-06-28 08:30:55.583 9999 pqr 5000.00 9999 pqr 4000.00

dbo DELETE 2021-06-28 08:31:25.940 NULL NULL NULL 9999 pqr 5000.00

=> display list of triggers ?

SELECT \* FROM sys.triggers

=> display trigger names and table names ?

SELECT t1.name,t2.name

FROM SYS.triggers T1,sys.tables T2

WHERE T1.parent\_id = T2.object\_id

Dynamic SQL :-

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

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

Example :-

1 DROP TABLE emp (Static SQL)

2 SET @tname ='EMP'

DROP TABLE @tname (Dynamic SQL)

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

=> Dynamic SQL commands are executed by using

1 EXEC command

2 sp\_executesql procedure

using EXEC command :-

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

=> dynamic sql command should be passed as string to EXEC

EXEC (' dynamic sql command')

Example 1 :-

DECLARE @tname varchar(10)

SET @tname ='EMP'

EXEC ('DROP TABLE ' + @tname)

Example 2 :-

create procedure to drop table ?

CREATE OR ALTER PROCEDURE drop\_table

@tname varchar(10)

AS

DECLARE @str VARCHAR(100)

SET @str = 'DROP TABLE ' + @tname

EXEC (@str)

Example 3 :-

create procedure to drop all tables ?

CREATE OR ALTER PROCEDURE drop\_all\_tables

AS

DECLARE C1 CURSOR FOR SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_TYPE='BASE TABLE'

DECLARE @tname VARCHAR(10),@str VARCHAR(100)

OPEN C1

FETCH NEXT FROM C1 INTO @tname

WHILE(@@fetch\_status=0)

BEGIN

SET @str = 'DROP TABLE ' + @tname

EXEC (@str)

FETCH NEXT FROM C1 INTO @tname

END

CLOSE C1

DEALLOCATE C1

write a prog to print no of rows in each and every table ?

EMP ?

DEPT ?

CUST ?

DECLARE C1 CURSOR FOR SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_TYPE='BASE TABLE'

DECLARE @tname VARCHAR(20),@str NVARCHAR(100), @cnt NVARCHAR(10)

OPEN C1

FETCH NEXT FROM C1 INTO @tname

WHILE(@@fetch\_status=0)

BEGIN

SET @str = 'SELECT @cntOUT=COUNT(\*) FROM ' + @tname

execute sp\_executesql @str,N'@cntOUT NVARCHAR(10) OUTPUT',@cntOUT=@cnt OUTPUT

PRINT @tname + ' ' + @cnt

FETCH NEXT FROM C1 INTO @tname

END

CLOSE C1

DEALLOCATE C1

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

03-JUL-21

EXISTS operator :-

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

=> used to check whether record exists in the table or not.

SELECT columns

FROM tabname

WHERE EXISTS (SELECT STATEMENT)

=> EXISTS returns TRUE / FALSE

TRUE => if subquery returns atleast one record

FALSE => if subquery returns 0 rows

scenario :-

-----------

PRODUCTS

PRODID PNAME PRICE

100

101

102

ORDERS

ORDID PRODID QTY

1000 100 1

1000 101 1

1001 100 1

=> display list of products ordered by customer ?

method 1 :-

SELECT \*

FROM products p

WHERE EXISTS (SELECT 1 FROM orders WHERE prodid = p.prodid)

100

101

method 2 :-

SELECT \*

FROM products p

WHERE prodid IN (SELECT prodid FROM orders)

=> SQL SERVER recommends EXISTS operator than IN operator because EXISTS gives

good performance than IN operator.

NOT EXISTS :-

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

=> display products which are not ordered by any customer ?

SELECT \*

FROM products p

WHERE NOT EXISTS (SELECT 1 FROM orders WHERE prodid = p.prodid)

102

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

export & import :-

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