**SQLSERVER SOFTWARE PATH:**

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

http://www.sqlschool.com/downloads

sqlserver

(rdbms)

(Microsoft Corportion)

**1.Operating system**

ex : Windows

**2.Application development tools**

ex: Visual studio (.net)

**3.Database server**

ex: Sqlserver

sql + TSQL => sqlserver developer => 3 weeks with project

msbi -> 4 weeks

**4.DWH**

ex: MSBI

ssis - Sqlserver integration services

ssrs - sqlserver reporting services

ssas - sqlserver analysis services

**5.ERP (Enterprise Resource planning)**

ex: Microsoft Dynamics

**Popular Rdbms products:**

**-----------------------**

**1.Sqlserver**

**2.Oracle**

**3.sybase**

**4.db2**

**5.mysql**

**6.ingres**

**7.Teradata**

**8.Msaccess**

**SQL : STRUCTURED QUERY LANGAUGE**

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

SQL WAS DEVELOPED BY IBM.

SQL is a common database language.

SQUARE : Specificiation of Query As Relational Expression

1973 : SEQUEL (Structured English Query language)

1979 : SQL (Structued query language)

SQL SUBLANGUAGE:

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

DDL - DATA DEFINATION LANGUAGE

DML - DATA MANIPULATION LANGUAGE

DCL - DATA CONTROL LANGUAGE

TCL - TRANSACTION CONTROL LANGUAGE

DQL - DATA QUERY LANGUAGE

**DDL - CREATE,ALTER,TRUNCATE,DROP**

**DML - INSERT,UPDATE,DELETE**

**DQL - SELECT**

**DCL - GRANT,REVOKE(SQLDBA)**

**TCL - COMMIT,ROLLBACK,SAVETRANSACTION**

**DATABASE OBJECTS:**

**-----------------**

**1.TABLE**

**2.VIEW**

**3.SYNONYM**

**4.INDEX**

**5.STORED PROCEDURE**

**6.STORED FUNCTION**

**7.STORED TRIGGER OR DATABASE TRIGGER**

**8.SEQUENCE (FROM SQLSERVER 2012)**

employee

ename empno salary (Fieldname)

aaa 1001 20000 Record /Tuple

bbb 1002 30000

ccc 1003 40000

ddd 1004 30000

kumar 1005 25000

jakeer 1006 35000

syed 1007 36000

raga 1008 45000

mano 1009 20000

Field/Attribute

**Data : collection of information.**

**Field : Each Column in a table is called as field or collection of related data is called field.**

**Record : Each row in a table is called Record or Collection of related fields are called Record.**

**Table : collection of related records are called Table.**

**Database : collection of related tables are called Database.**

**DBMS : It is a software used to insert,update,delete the records in single table.**

**RDBMS : It is a software used to insert,update,delete the records in related table.**

**SQL : STRUCTURED QUERY LANGAUGE**

**DDL : CREATE,ALTER,TRUNCATE,DROP**

**DML : INSERT,UPDATE,DELETE**

**DQL : SELECT**

**TCL : COMMIT,ROLLBACK,SAVE TRANSACTION**

**DCL : GRANT,REVOKE(SQLDBA)**

**File:**

-----

* A file is a collection of data stored in one unit, identified by a filename.
* It can be a document, picture, audio or video stream, data library, application, or other collection of data.

**File System:**

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

* A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition;
* This is the way the files are organized on the disk.

**Disadvantages of the file system:**

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

1. Data Redundancy
2. Difficulty in Accessing Data
3. Limited Data Sharing
4. Integrity Problems
5. Concurrent Access
6. Security Problems

**Database :**

* A database is an collection of data that is organized to be easily accessed, managed and updated to relational databases.

**Database Model:**

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

* A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized and manipulated.

The most popular example of a database model is the relational model, which uses a table-based format.

**Types of database models**

1. 1.Hierarchical database model.
2. 2.Relational model.
3. 3.Network model.
4. 4.Object-oriented database model.

**Hierarchical Database Model:**

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

* A Hierarchical database model is a data model in which the data are organized into a tree-like structure.
* The data are stored as records which are connected to one another through links. A record is a collection of fields, with each field containing only one value.

ex: hierarchical databases is Windows Registry in the Microsoft Windows operating systems.

**Network Database Model:**

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

Network database model is a model for modeling the entities in such a way that one child entity can have more than one parent entities.

**Characteristics of network model:**

**----------------------------------**

* More better than hierarchical model.
* Supports many to many relationships.
* Many parent can have many child.
* Many child can have many parents.
* Complex structure.
* Not very flexible to reorganize the model.
* Query facility is not available in network model.

**Object Oriented Database Model:**

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

* Object Oriented Database (OODB) provides all the facilities associated with object oriented paradigm.
* It enables us to create classes, organize objects, structure an inheritance hierarchy and call methods of other classes.

In sqlserver 2017 built in databases:

1.master

2.model

3.msdb

4.tempdb

5.resource [ read only & hidden ]

**System databases :**

System databases contains system-tables,system stored procedure and system level information.

**Master :**

----------

* It contains information about Server's configuration.
* Master database stores logins information and addresses of all the databases,System stored procedures,system-tables,system stored function and system stored triggers etc...
* The database id is 1.
* Core system database to manage Sql Server instance.
* this is the heart of the sqlserver,Without Master database, server can't be started.

**Model :** Model is a template databases. Every user defined databases inherits the default strucutre from model database.

* The database id is 3.

**MSDB :(Microsoft Database**)

MSDB database stores jobs,schedules, operators,alert,,Logshiping,Database Mail configuration,Backup and Restore,Maintainance plan operation etc...

The database id is 4.

select \* from sys.databases

**Tempdb** : It is temporary database, database engine allocates temporary memory in this databases.

* \*This database id is 2.

why we need to create the temporary tables?

grouping ,sorting,filtering,joining this cases using temporary memory.

**faq:**

-------

what to do when tempdb is full.

restart the server where downtime is possible.

**Resource :** [ hidden and readonly ]

It stores system related objects,service pack information etc.

\* The database id 32767.

**Note:**

Backup can be taken for all system databases except "tempdb" database.

**faq:**

what to do when tempdb is full.

restart the server where downtime is possible.

VERSIONS IN SQL-SERVER:

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

1.0 (1989)

3.0

4.0/4.5

6.0/6.5

7.0

8.0 (SQL-SERVER 2000) (major version no.minor version no)

9.0 (SQL-SERVER 2005)

10.0 (SQL-SERVER 2008 RTM)

10.50 sqlserver 2008 R2)

11.0 sqlserver 2012

12.0 sqlserver 2014

13.0 Sqlserver 2016

14.0 sqlserver 2017

editions of sql server 2014:

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

**(i) sqlserver datacenter:**

-- introduced from sqlserver 2008 R2

-- recommended for very large size application

-- supports all the features like replication,log shipping,db mirroring,clustering,data partitioning, index partitioning etc

-- supports clustering upto a maximumn of 16 nodes.

-- no limit for no. of CPU's

-- no limit for RAM size and database size.

**(ii) ENTERPRISE EDITION:**

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

-- recommended for large scale applications.

-- support all the features

--maximum of cpu's is 8

-- maximum RAM size is 2TB

-- no limit for the database size

-- supports clustering upto a max of 16 nodes.

**(iii) STANDARD EDITION:**

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

-- recommended for medium scale applications

-- supports clustering upto a maximum of 2 nodes

-- maximum number of CPU's is 4

-- maximum RAM size is 64GB

-- no limit for Database size

**(iv) web edition:**

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

-- recommended for web based applications

--does not supports clustering

-- max number of cpu's is 4

-- max size of RAM is 64 GB

-- no limit for DB size

**(v) WORKGROUP EDITION**:

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

-- recommeded for small scale application where a group of users are working

-- does not supports clustering

-- max number of CPU's is 2

-- max size of RAM is 4GB

-- no limit for DB size

**(vi) DEVELOPER EDITION:**

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

--supports all the features of datacenter edition

-- recommended only for developement or testing purpose

**(vii) EXPRESS EDITION**

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

-- a freely downloadable edition with only minimal features

-- doesn't not support any high availability features,automation etc

-- max number of CPU's is 1

-- max RAM size 1GB

-- max size of Database is 10GB

--how to open sqlserver?

start -> all programs -> sqlserver 2017 -> sqlserver management studio

or

start -> run -> ssms

--How to create a new database?

create database <databasename>

ex:

create database Shruthi

--how to open a database?

**use <databasename>**

use Shruthi

create database Shruthi

use Shruthi

**select db\_name()**

**select db\_id()**

**select host\_name()**

**select host\_id()**

**select suser\_name()**

**select sUSER\_Id()**

**select @@version --Global variable**

**SQLSERVER OPERATORS:**

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

**Arithmetical operators:**

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

+ addtion

- subtract

\* multiply

/ division (quotient)

% division (remainder)

**Relational operators or comparision operators:**

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

< less than

> greater than

<= less than or equal to

>=greater than or equal to

= equal to

!= not equal to

**Logical operators:**

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

and

or

not

and

a b c=a\*b

0 0 0

0 1 0

1 0 0

1 1 1

or

a b c=a+b

0 0 0

0 1 1

1 0 1

1 1 1

not

a c

0 1

1 0

**set operators:**

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

union

union all

intersect

except

A B

{101,102,103,104,105} union {101,103,105,107,109}

=>{101,102,103,104,105,107,109}

{101,102,103,104,105} union all {101,103,105,107,109}

=>{101,102,103,104,105,101,103,105,107,109}

{101,102,103,104,105} Intersect {101,103,105,107,109}

=>{101,103,105}

{101,102,103,104,105} except {101,103,105,107,109}

A-B =>{102,104}

B-A =>{107,109}

hyd

ename eno salary

aaa 101 2000

bbb 102 3000

ccc 103 4000

ddd 104 5000

eee 105 6000

che

name eno sal

xxx 101 3500

yyy 103 4000

zzz 105 5000

mmm 107 3500

nnn 109 4500

create table hyd(ename varchar(20),eno int,salary int)

insert into hyd values('aaa',101,2000),('bbb',102,3000),('ccc',103,4000),('ddd',104,5000),('eee',105,6000)

create table che(name varchar(20),eno int,sal int)

insert into che values('xxx',101,3500),('yyy',103,4000),('zzz',105,5000),('mmm',107,3500),('nnn',109,4500)

--example1

select eno from emp1

union

select eno from emp2

--example 2

select eno from emp1

union all

select eno from emp2

--example3

select eno from emp1

Intersect

select eno from emp2

--example 4

select eno from emp1

except

select eno from emp2

--example 5

select eno from emp2

except

select eno from emp1

**Sqlserver Field types or datatypes:**

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

**Character Datatypes:**

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

char(40) - Fixed length char - Ascii - 1 byte - Static memory allocation

Varchar(40) - Variant length char - Ascii - 1 byte - Dynamic memory allocation

nchar(40) - Fixed length char - unicode - 2 bytes - Static memory allocation

nvarchar(40) - Variant lentgh char - unicode - 2 bytes

**Dynamic memory allocation**

**Number datatypes:**

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

tiny int -1 byte - 8 bits - 2 power 8 => 256/2=> 128

-128 to +127

small int -2 bytes - 16 bits - 2 power 16 => 65536/2

32,768 => -32,768 to +32,767

int - 4 bytes => 32 bits => 2 power 32

big int - 8 bytes = 64 bits => 2 power 64

**Decimal point datatypes:**

**------------------------**

float - 4 bytes (Approx. 7 decimal digits)

real - 8 bytes (approx. 14 decimal digits)

**Datetime datatypes:**

**------------------**

smalldatetime - 4 bytes - 01-01-1900 to 06-06-2078

datetime - 8 bytes -> 01-01-1753 to 31-12-9999

**Currency datatypes:**

**-------------------**

small money - 4 bytes

money - 8 bytes

**xml datatypes: (Extensible markup language)**

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

* xml
* image

**DDL - CREATE,ALTER,TRUNCATE,DROP**

**--How to create a new table?**

**syntax:**

**create table <tablename>(Fieldname1 fieldtype1,fieldname2 fieldtype2,fieldname3 fieldtype3,.....)**

--ex:

create table employee(ename varchar(20),empno int,salary int)

**--How to insert the records?**

**syntax:**

**insert into <tablename> values('value1',value2,value3)**

ex:

insert into employee values('aaa',101,25000)

insert into employee values('bbb',102,35000)

insert into employee values('ccc',103,45000)

**--How to select the records?**

**syntax: (user follows)**

**select <fieldname>**

**from <tablename>**

**where <condition>**

**group by <Fieldname>**

**having <condition>**

**order by <fieldname> asc/desc**

**--System follows**

from <tablename>

where <condition>

group by <Fieldname>

having <condition>

order by <fieldname> asc/desc

select <fieldname>

create table employee(ename varchar(20),empno int,salary int)

insert into employee values('aaa',101,25000)

insert into employee values('bbb',102,35000)

insert into employee values('ccc',103,45000)

select \* from employee

select ename,empno from employee

select ename,salary from employee

select \* from employee where empno=101

student

sname sno mark

aaa 101 100

bbb 102 200

ccc 103 250

create table student(sname varchar(20),sno int,mark int)

insert into student values('aaa',101,100)

insert into student values('bbb',102,200)

insert into student values('ccc',103,250)

select \* from student

**alter command:**

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

* is used to adding a new column in a table.
* is used to modify existing column in a table(name,size).
* is used to delete particluar column in a table.

**Adding a new Field syntax:**

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

**alter table <Tablename> add <fieldname> <fieldtype>(size)**

ex:

alter table employee add hra int

**what is meant by Null?**

* Null is not equal to zero.
* Null is not equal to empty.
* Null is not equal to blankspace.
* null is not equal no values.
* null is an undefined value.

**Note : Each and every table we have 2 types of data**

* Actual data => select \* from tablename
* Meta data => data about data => sp\_help 'tablename'

(stored procedure)

**--How to modify the column size?**

**syntax:**

**alter table <Tablename> alter column <fieldname> <fieldtype>(size)**

ex:

alter table employee alter column ename varchar(30)

verification

**sp\_help 'employee'**

**--how to delete particular column in a table?**

syntax:

**alter table <Tablename> drop column <fieldname>**

ex:

alter table employee drop column hra

alter table employee drop column hra

select \* from employee

--list all ename,empno,sal,comm,total salary from emp table

select ename,empno,sal,comm,sal+isnull(comm,0) as total from emp

--How to create a new database?

create database Madhu

--How to open database?

use Madhu

create table employee(ename varchar(20),empno int,salary int)

insert into employee values('aaa',101,25000)

insert into employee values('bbb',102,35000)

insert into employee values('ccc',103,45000)

select \* From employee

**Truncate command:**

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

* is used to delete all records permanently.
* it cannot be rollback.

**syntax:**

**truncate table <Tablename>;**

ex:

truncate table employee;

--verify

select \* from employee;

**drop command:**

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

* is used to delete whole table or whole object permanently.
* it cannot be rollback.

**syntax:**

**drop table <Tablename>**

ex:

drop table employee

--verify

select \* from employee;

--List all emp details

select \* from emp

--list all ename,empno,sal from emp table

select ename,empno,sal from emp

--list all emp details who are working with deptno 10

select \* From emp where deptno=10

--List all emp details whose earning salary above 2500

select \* From emp where sal >=2500

--list all emp details who are joined in the year 1981.

select \* from emp where hiredate >='1981-01-01' and hiredate <='1981-12-31'

--list all emp details who are working with deptno 10 and 30.

select \* from emp where deptno=10 or deptno=30

--list all emp details who are working as a clerk,manager and salesman

select \* from emp where job='clerk' or job='manager' or job='salesman'

**isnull(exp1,exp2):**

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

* Is null function are used to convert null values into some other values.
* it accept 2 parameters.
* if exp1 is null return exp2 else exp1.

--List all ename,empno,daily,monthly and ctc from emp table

select ename,empno,sal /30 as daily,sal as monthly,sal \* 12 as ctc from emp

--List all ename,empno,sal,comm ,totalfrom emp table

select ename,empno,sal,comm ,sal + isnull(comm,0) as total from emp

--list only king details

select \* From emp where ename='king'

--list all ename,empno,sal,hra 10% ,ma 12%,ta 15% of basicpay

select ename,empno,sal,sal \* 10/100 as hra,sal \* 12/100 as ma,sal \* 15/100 as ta

from emp

**DML - INSERT,UPDATE,DELETE**

**Insert command syntax:**

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

1.insert [into] <tablename> values('value1',value2,value3') --all values

2.insert [into] <tablename> (colname1,colname2) values('value1',value2) --parital values

3.insert [into] <tablename> values('value1',value2,value3),('value1',value2,value3),('value1',value2,value3),

('value1',value2,value3),('value1',value2,value3),...... (multiple records)

insert command syntax:

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

**I method (only one record all values)**

insert into <tablename> values('value1',value2,value3)

**II method (only one record partial values)**

insert into <tablename>(colname1,colname2) values('value1',value2)

**III method (Multiple records)**

insert into <Tablename> values('value1',value2,value3),(value1,value2,value3),(value1,value2,value3)

**update command:**

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

* is used to modify the record values.

syntax:

**update <Tablename> set <fieldname>=value where <condition>**

alter table dept add TL varchar(10)

select \* from dept

UPDATE DEPT SET TL='SHARMA' WHERE DEPTNO=10

UPDATE DEPT SET TL='RAGHU' WHERE DEPTNO=20

UPDATE DEPT SET TL='SHEKAR' WHERE DEPTNO IN(30,40)

SELECT \* FROM DEPT

IF DEPTNO=10 THEN SAL+100

IF DEPTNO=20 THEN SAL+200

IF DEPTNO=30 THEN SAL+300

UPDATE EMP SET SAL=SAL+100 WHERE DEPTNO=10

SELECT \* FROM EMP

UPDATE EMP SET SAL=SAL+200 WHERE DEPTNO=20

UPDATE EMP SET SAL=SAL+300 WHERE DEPTNO=30

SELECT \* FROM EMP

--HOW TO UPDATE MULTIPLE CONDITIONAL VALUES IN SINGLE UPDATE STATEMENT? (FAQ)

UPDATE <TABLENAME> SET FIELDNAME=CASE <VARIABLE>

WHEN 1 THEN 'STATEMENT1'

WHEN 2 THEN 'STATEMENT2'

WHEN 3 THEN 'STATEMENT3'

ELSE 'INVALID STATEMENT'

END

--EXAMPLE

UPDATE EMP SET SAL=CASE DEPTNO

WHEN 10 THEN SAL+100

WHEN 20 THEN SAL+200

WHEN 30 THEN SAL+300

ELSE SAL+0

END

**--DELETE COMMAND:**

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

* Is used to delete single records or group of records or all records temporarly.
* it can be rollback.

syntax:

**delete from <tablename> where <condition>**

ex:

delete from emp where sal >=2500

delete from emp where deptno=10

delete from emp where job='clerk'

delete from emp where comm is null

delete from emp; -- all records

delete from emp where sal>=2500

select \* from emp

delete from emp where deptno=20

select \* from emp

delete from emp where job='salesman'

select \* from emp

delete from emp

**What is the difference between Truncate and Delete command?**

**Differences between Delete and Truncate Commands;**

**----------------------------------------------------------------------**

**DELETE TRUNCATE**

**----------------------------------------------------------------------**

**1. It is DML command. 1. It is DDL command.**

**2. Deletes rows for Temporary 2. Deletes rows for permanent**

**3. Supports to delete rows on 3. No conditional deletion**

**conditional basis**

**4. It is slow in execution,since 4. It is fast in execution**

**it consults log file to store since it does not**

**each deleted row consult log.**

**5. When all rows are deleted, 5. When all rows are**

**identity column value still deleted, identity col**

**persist. value will be set to 1**

**----------------------------------------------------------------**

**Transaction control language:**

**-----------------------------**

* Transaction is nothing but a logical unit of work.
* (insert,update,delete).

**Commit** - To make the transaction permanently.

**Rollback -** To cancel the transaction.

**SAVETRANSACTION** - is used to divide the large transaction into small transaction.

**Types of Transaction :**

**---------------------**

* **1.Implicit Transaction (Auto commit mode)**
* **2.Explicit Transaction (User follows)**

--Example1

Begin Transaction

select \* from emp -- 14 rows

delete from emp

select \* from emp -- 0 rows

rollback transaction

select \* From emp --14 rows

--example2

begin transaction

select \* from dept --4 rows

delete from dept where deptno=40

select \* from dept --3 rows

commit transaction

select \* from dept --3 rows

--example3

Begin transaction

create table e1(ename varchar(20))

save transaction st1

insert into e1 values('aaa'),('bbb'),('ccc')

save transaction st2

insert into e1 values('aaa'),('bbb'),('ccc')

save transaction st3

insert into e1 values('aaa'),('bbb'),('ccc')

select \* from e1 -- 9 rows

rollback transaction st3

select \* from e1 -- 9 rows

rollback transaction st2

select \* from e1 -- 3 rows

commit transaction

select \* from e1

**Sql server Special operators:**

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

in

not in

between

not between

like

not like

is null

is not null

exists

not exists

**In operators are used to select multiple values or more than one values or particluar values.**

**Not in operators are used to select except multiple values or except more than one values or except particular values.**

**syntax:**

**select \* from tablename where <fieldname> in (value1,value2,value3)**

select \* from tablename where <fieldname> not in

(value1,value2,value3)

--list all emp details who are working with deptno 10 and 30.

select \* from emp where deptno in(10,30)

--except deptno 10 and 30.

select \* from emp where deptno not in(10,30)

--List all emp details who are working as clerk,manager,president

select \* from emp where job in('clerk','manager','president')

--except clerk,manager and president

select \* from emp where job not in('clerk','manager','president')

--list all emp details whose earning salary equal to 3000 and 5000.

select \* from emp where sal in(3000,5000)

**Between and not between operators:**

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

* between operators are used to select range of values.
* Not between operators are used to select except particluar range of values.
* only supports number and datetime datatypes.

**syntax:**

select \* from tablename where <fieldname> between <lowestvalue> and <highestvalue>

select \* from tablename where <fieldname> not between

<lowestvalue> and <highestvalue>

--list all emp details whose earning salary between 2000 to 3000.

select \* from emp where sal between 2000 and 3000

--Except 2000 to 3000

select \* from emp where sal not between 2000 and 3000

--list all emp details who are joined in the year 1981

select \* from emp where hiredate between '1981-01-01' and '1981-12-31'

--except 1981

select \* from emp where hiredate not between '1981-01-01' and '1981-12-31'

**Like and not like operators:**

**--------------------------**

\* Are used to select Pattern matching characters.

\* we have 3 special characters

% - multi character

\_ - single character

[] - Range of characters

--List all emp details whose ename starts with 'S' letter

select \* from emp where ename like 'S%'

--ends with S letter

select \* from emp where ename like '%S'

--starts with A and J Letter

select \* from emp where ename like 'A%' or ename like 'J%'

--starts with A to J

select \* from emp where ename like '[A-j]%'

--except A to j

select \* from emp where ename not like '[A-J]%'

--List ALL EMP DETIALS WHOSE ENAME LENGTH OF CHAR IS 4.

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

--DISPLAY ALL EMP DETAILS WHOSE ENAME STARTS WTIH 3RD CHAR A LETTER.

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

--LIST ALL EMP DETAILS WHOSE EMPNO STARTS WITH '78%'

SELECT \* FROM EMP WHERE EMPNO LIKE '78%'

**IS NULL AND IS NOT NULL OPERATORS:**

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

* IS null operator is used to select null values.
* is not null operators are used to select except null values.

--LIST ALL EMP DETAILS WHOSE EARNING COMMISSION

SELECT \* FROM EMP WHERE COMM IS NOT NULL

--LIST ALL EMP DETAILS WHOSE NOT EARNING COMMISSION

SELECT \* FROM EMP WHERE COMM IS NULL

**Exists and not exists operators:**

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

* Are used to check the availalbity.
* it returns true or false.

if not exists(select deptno from dept where deptno=50)

insert into dept values(50,'HR','HYD')

select \* from dept

if exists(select deptno from dept where deptno=50)

delete from dept where deptno=50

**SQL FUNCTION**

**1.GROUP FUNCTION / AGGREGATE FUNCTION**

A.SUM()

B.AVG()

C.MAX()

D.MIN()

E.COUNT()

F.COUNT\_BIG()

G.OVER()

**Note : These functions are execute group of records and return only one value.**

**2.SINGLE ROW FUNCTION**

A.STRING FUNCTION

B.MATHEMATICAL FUNCTION

C.DATETIME FUNCTION

D.CONVERSION FUNCTION

E.SPECIAL FUNCTION

F.OLAP FUNCTION (ONLINE ANALYTICAL PROCESSING)

A.RANK() B.DENSE\_RANK() C.ROW\_NUMBER()

**Note : These functions are execute only one record and return only one value.**

select count(sal) as count from emp -- it does not count null values

select count(\*) as count from emp --it count null values

select count(comm) as count from emp --int 4 byte

select count\_big(sal) as count\_big from emp -- big int 8 bytes

select sum(sal) as sum from emp

select avg(sal) as avg from emp

select max(sal) as max from emp

select min(sal) as min from emp

select count(sal) as count from emp

GROUP BY CLAUSE:

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

1.IT IS USED TO CREATE GROUP RECORDS BASED ON ONE OR MORE COLUMNS TO CALCULATE AGGREGATES LIKE MAX,MIN,SUM,AVG,COUNT ETC.....

2.USING GROUP BY WE CAN PRODUCE SUMMAIRIZED DATA FROM DETAILED DATA.

3.The GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

4.select command is used to filter the data.

The syntax for the GROUP BY clause is:

SELECT column1, aggregate\_function (expression)

FROM table\_NAME

WHERE condition

GROUP BY column1

--display sum of salary for each dept

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

--display avg.salary for each dept

select deptno,avg(sal) as avg from emp group by deptno

--display max.salary for each dept

select deptno,max(sal) as max from emp group by deptno

--display min.salary for each dept

select deptno,min(sal) as min from emp group by deptno

--display no.of employees working in each dept

select deptno,count(\*) as count from emp group by deptno

--display no.of employees working in each job wise

select job,count(\*) as count from emp group by job

rollup clause:

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

\* is used to calculate aggregate with aggregate.

\* Rollup clause can be used in a Group by clause.

select deptno,sum(sal) as sum,avg(sal) as avg,max(sal) as max,

min(sal) as min,count(sal) as count from emp group by deptno with rollup

select deptno,sum(sal) as sum,avg(sal) as avg,max(sal) as max,min(sal) as min,

count(sal) as count from emp group by deptno with rollup

HAVING CLAUSE:

Having clause is used with the select clause to specify a search condition for a group or aggregate.

What is the difference between where clause and having clause?

1. 'Where' clause applies to the individual rows whereas 'Having' clause is used to test some condition on the group(usually aggregate methods) ...

2.To filter data before group by use where clause.

3.To filter data after group by use having clause.

4.In condition,if there is no aggregate function then use where clause.

5.In condition, if there is aggregate function,then use having clause.

--display no.of employees working in each dept and no.of employees >=5

select deptno,count(\*) as count from emp where count(\*) >=5 group by deptno --invalid

select deptno,count(\*) as count from emp group by deptno having count(\*)>=5 --valid

--list sum of salary for each year wise and month wise

select datename(mm,hiredate) +'-'+datename(yyyy,year(hiredate) ),sum(Sal) as sum from emp

group by datename(mm,hiredate) ,year(hiredate)

--display sum of salary for dept 10 and 30

select deptno,sum(sal) as sum from emp where deptno in(10,30) group by deptno

--or

select deptno,sum(sal) as sum from emp group by deptno having deptno in(10,30)

--display no.of employees working in each dept and no.of employees >=4

select deptno,count(\*) as count from emp group by deptno having count(\*)>=4

--display no.of employees joined in each year

select year(hiredate) as year,count(\*) as count from emp group by year(hiredate) with rollup

--display no.of employees joined in the year 1981 and 1983

select year(hiredate) as year,count(\*) as count from emp

where year(hiredate) in(1981,1983) group by year(hiredate) with rollup

--or

select year(hiredate) as year,count(\*) as count from emp

group by year(hiredate) having year(hiredate) in(1981,1983)

1.Group by Clause --group records

2.Rollup clause -- aggregate with aggregate

3.where clause -- To filter the data before group by & 4 th point

4.having clause --To filter the data after group by &

5.Order by clause --Arrage the data either asc or desc.

6.Distinct clause --Eliminate the duplicate value.

7.Identity Clause --Auto number

ORDER BY CLAUSE :

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

IT IS USED TO arrange records either ascending order or descending order.

by default ascending order

ASC

Specifies that the values in the specified column should be sorted in ascending order, from lowest value to highest value.

DESC

Specifies that the values in the specified column should be sorted in descending order, from highest value to lowest value.

syntax:

SELECT column\_name

FROM table\_name

[WHERE condition]

\*ORDER BY column\_name [ASC, DESC]

select \* from emp order by deptno asc

select \* from emp order by sal desc

select \* from emp order by deptno asc,sal desc

select \* from emp order by 8 asc,6 desc

select ename,empno,sal,deptno from emp order by 4 asc,3 desc

--List out the employee id, ename in ascending order based on the employee id.

select empno,ename from emp order by empno

--list out all emp details,deptwise ascending order

select \* from emp order by deptno asc

--salary wise desc order

select \* from emp order by sal desc

--dept wise asc order and salary wise desc order

select \* from emp order by deptno asc,sal desc

--or

select \* from emp order by 8 asc,6 desc

--or

select ename,empno,sal,deptno from emp order by 4 asc,3 desc

select \* from emp order by ename asc

--display all emp details in empno wise asc order

select \* from emp order by empno asc

--display all emp details in deptno wise desc order

select \* from emp order by deptno desc

--deptno asc,sal wise desc

select \* from emp order by deptno asc,sal desc

select \* from emp order by 8 asc,6 desc

select ename,empno,sal,deptno from emp order by 4 asc,3 desc

Distinct clause/ Distinct Operators:

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

\* is used to eliminate or remove the duplicate values.

syntax

select distinct (colname) from tablename

or

select distinct colname from tablename

--how to know howmany dept are available?

select distinct (deptno) from emp

select distinct job from emp

select distinct mgr from emp where mgr is not null

--how many dept are created?

select count(distinct deptno) from emp

select distinct (deptno) from emp

--how many depts are available?

select count(distinct Deptno) from emp

select distinct mgr from emp where mgr is not null

select distinct job from emp

Identity clause:

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

\* is used to generate the sequential integer values.

\* it automatically generates the primary key or unique key values.

\* it can be either asc order or desc order.

ex:

create table emp11(ename varchar(20),eno int identity(100,1),salary int)

100 -> starting identity value

1-> increment identity value

(1,1) -> by default

insert into emp11 values('aaa',25000)

insert into emp11 values('bbb',35000)

insert into emp11 values('ccc',35000)

insert into emp11 values('ddd',2500)

select \* from emp11

ename eno salary

aaa 100 25000

bbb 101 35000

ccc 102 35000

ddd 103 2500

select ident\_seed('emp11')

select IDEnt\_incr('emp11')

select IDENT\_CURRENT('emp11')

select @@identity --CURRENT IDENTITY VALUES

select scope\_identity() -- it returns current identity values

Single row function:

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

1.String function

2.Mathematical function

3.datetime function

4.conversion function

5.special function

6.OLAP Function

Note: These functions are execute only one row and return only one value.

String function

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

select len('anusha')

select len('anusha hyd')

select upper('anusha')

select lower('syed')

select reverse('anusha')

select replicate('anusha ',5)

select left('anusha syed',6)

select right('anusha syed',5)

select substring('have a nice day',6,6)

select substring('have a nice day',8,4)

select replace('anusha syed','syed','ibm')

select ascii('A')

select char(65)

select charindex('x','have a nice day')

select ascii('A')

select char(65)

select replace('kamal sqlschool','sqlschool','ibm')

select charindex('x','kamal')

select charindex('al','kamal')

Mathematical function

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

select abs(-4994)

select sin(30)

select cos(30)

select tan(30)

select log(10)

select exp(10)

select power(10,3)

select square(100)

select sqrt(100)

select sign(-4994)

select sign(4004)

select sign(0)

select round(8888.8888,2)

select round(8888.8888,1)

select round(8888.8888,0)

select round(8888.8888,-1)

select round(8888.8888,-2)

select ceiling(8888.8888)

select floor(8888.8888)

select pi()

Datetime function:

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

select getdate()

select year(getdate())

select month(getdate())

select datename(mm,getdate())

dateformat

dd - day

dw - day of week

mm - month

yy - year

hh - hour

mi - minute

ss - seconds

qq - quarter of year

ww - week of year

select datename(dw,getdate())

select datepart(dw,getdate())

select getdate() + 100

select dateadd(dd,100,getdate())

select dateadd(mm,100,getdate())

select dateadd(hh,100,getdate())

select datediff(dd,getdate(),'2016-03-25')

select datediff(hh,getdate(),'2016-03-25')

select datediff(yy,'1993-10-25',getdate())

select eomonth(getdate())

select DATEPART(dd,EOMONTH(getdate()))

3.--Calculating Age in years, months and days

--Write a tsql program to get the years, months and days between two dates.

Declare @dateofbirth datetime

Declare @currentdatetime datetime

Declare @years varchar(40)

Declare @months varchar(30)

Declare @days varchar(30)

set @dateofbirth='2016-12-06'--birthdate

set @currentdatetime =getdate()--current datetime

select @years=datediff(year,@dateofbirth,@currentdatetime)-- To find Years

select @months=datediff(month,@dateofbirth,@currentdatetime)-(datediff(year,@dateofbirth,@currentdatetime)\*12)

-- To Find Months

select @days= datepart(d,@dateofbirth) - datepart(d,@currentdatetime)-- To Find Days

select @years +' years, ' +@months +' months, '+@days +' days' asYearMonthDay

first day of the month

SELECT DATEADD(dd,-(DAY(GETDATE())-1),GETDATE())

Conversion function

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

\* These functions are used to convert the one datatype into another datatype.

char -> int

int -> char

date -> char

datetime ->date

datetime -> time

Types:

--------

cast() -> ansi syntax --support all rdbms

convert() -> sqlserver syntax - only support sqlserver

cast() syntax:

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

cast(expression as datatype)

select cast(1111.55555 as int) -> 1111

select cast(1111.55555 as int)

select cast(getdate() as varchar)

select cast('2015-march-25' as datetime)

select ename , '$'+ str(cast(sal as money))+'.000' as sal from emp

convert() syntax:

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

convert(datatype,expression)

select convert(int,1111.555) ->1111

select convert(varchar,getdate())

select convert(varchar,getdate(),101)

select convert(varchar,getdate(),102)

select convert(varchar,getdate(),103)

select convert(varchar,getdate(),104)

select convert(varchar,getdate(),105)

select convert(varchar,getdate(),131)

https://www.w3schools.com/sql/func\_sqlserver\_convert.asp

--find the percentage of emp salary and 2 decimal point

select ename,empno,convert(varchar,convert(money,sal)) +'%' as sal from emp

--how to display reverse words? (stored function)

special function:

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

select db\_name()

select db\_id()

select suser\_name()

select host\_name()

select ident\_seed('tablename')

select ident\_incr('tablename')

select ident\_current('tablename')

OLAP Function : (online analytical processing)

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

rank() -- it generates gap

dense\_rank() -- it does not generates gap.

row\_number() -- is used to generate sequential integer value.

select ename,empno,sal ,rank() over(order by sal desc) as rnk from emp

select ename,empno,sal,dense\_rank() over(order by sal desc) as dense from emp

select ename,empno,sal,row\_number() over(order by sal desc) as rn from emp

rank dense\_rank() row\_number()

1 1 1

1 1 2

3 2 3

3 2 4

5 3 5

6 4 6

7 5 7

7 5 8

7 5 9

10 6 10

10 7 11

12 8 12

13 8 13

14 9 14

--list all ename,empno,sal ,rank from emp table

select ename,empno,sal,dense\_rank() over(order by sal desc) as dense from emp

select ename,empno,sal,row\_number() over(order by sal desc) as rn from emp

select ename,empno,sal,rank() over(order by sal desc) as rnk,dense\_rank() over(order by sal desc) as dense,

row\_number() over(order by sal desc) as rn from emp

Constraints: (Condition)

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

train\_master

train\_no train\_name

2655 apexpress

2655 charminarexp

2655 chennaiexpress

2656 apexpress

student

sname sno mark (0 to 100)

emp

ename empno(pk) gender(m/f) doj (m to f) salary (20000 to 50000) pan int (10 digits)

Constraints:

-----------

1.it is used to controlling the data in a table.

2.it is used to prevent the invalid data entry into table.

3.it also used to maintain the security of the table.

4.it is used to implement the business rules.

or

The Process or enforcing the business rules on database tables is called "Data Integrity".

There are two types of data integrity:

1.Predefined integrity [constraints]

2.User defined integrity [ Database triggers]

Predefined Integrity:

a.Domain integrity constraints

b.Entity Integrity Constraints

c.Referentital Integrity constraints

Domain Inegrity Constraints:

a.Check constraints

b.not null constraints

c.Default constraints

Entity Integrity constraints

a.Primary key constraints

b.unique key constraints

Referential Integrity Constraints

a.Foreign key constraints

Check constraints:

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

\* is used to validate the data based on the condition.

salary -> 20000 to 50000

gender -> m/f

mark -> 0 to 100

doj -> monday to friday

pan -> 10 digits

Not null constraints:

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

\* it does not allow null values.

\* it accept duplicate values.

Default constraints:

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

\* Sqlserver insert some default values.

PRIMARY KEY CONSTRAINTS:

1.A COLUMN DECLARED WITH PRIMARY KEY DOESNOT ALLOW DUPLICATES AND NULL VALUES.

2.A COLUMN DECLARED WITH PRIMARY KEY, UNIQUELY IDENTIFIES EACH RECORD IN A TABLE.

3.A TABLE ALLOWS ONLY ONE PRIMARY KEY.

4. NUMERIC COLUMNS ARE PREFERED AS PRIMARY KEYS THAN CHARACTER COLUMNS.

UNIQUE KEY CONSTRAINTS:

1.UNIQUE DOESNOT ALLOW DUPLICATE VALUES.

2.it accepts only one null values.

3.A table allows more than one unique key.

student

sname sno mark(0 to 100)

create table student(sname varchar(20),sno int,mark int check(mark between 0 and 100))

insert student values('anusha',101,99) --valid

insert student values('bbb',102,100) --valid

insert student values('ccc',103,101) --invalid

select \* from student

emp1

ename empno(pk) gender(m/f) salary (20000 to 50000)

drop table emp1;

create table emp1(ename varchar(20),empno int primary key,gender char(1)

check(gender in('m','f')),salary int check(salary between 20000 and 50000))

insert emp1 values('aaa',1001,'m',25000) --valid

insert emp1 values('bbb',1001,'f',35000) --invalid

insert emp1 values('ccc',null,'f',35000) --invalid

insert emp1 values('ddd',1002,'x',30000) --invalid

insert emp1 values('eee',1003,'f',15000) --invalid

select \* from emp1

Not null constraints:

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

\* it does not allow null values.

\* it accept duplicate values.

create table emp2(ename varchar(20),eno int not null,salary int)

insert emp2 values('aaa',101,2000) --valid

insert emp2 values('bbb',102,3000) --valid

insert emp2 values('ccc',101,3000) --valid

insert emp2 values('ddd',null,3900) --invalid

select \* from emp2

unique key constraints:

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

create table emp3(ename varchar(20),eno int unique,salary int)

insert emp3 values('aaa',101,2500) --valid

insert emp3 values('bbb',101,3000) --invalid

insert emp3 values('ccc',null,5000) --valid

insert emp3 values('ddd',null,5000) --invalid

select \* from emp3

create table train\_master(train\_name varchar(20) unique,train\_no int unique)

insert into train\_master values('Ap Express',2655) --valid

insert into train\_master values('Chennai express',2656) --valid

insert into train\_master values('Ap Express',2657)--invalid

default constriants:

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

\* Sqlserver insert some default values

create table emp\_entry(eid int,in\_time datetime default getdate())

insert into emp\_entry(eid) values(1001)

insert into emp\_entry(eid) values(1003)

insert into emp\_entry(eid) values(2001)

insert into emp\_entry(eid) values(2002)

select \* from emp\_entry

create table emp5(ename varchar(20),

eid int, org varchar(20) default 'unisys',

salary int)

insert emp5(ename,eid,salary)

values('ccc',1003,4500)

select \* From emp5

order\_master

orderdate orderno

2015-03-25 1

2015-03-25 2

2015-03-25 3

2015-03-25 4

2015-03-26 1

2015-03-26 2

2015-03-27 1

2015-03-27 2

2015-03-27 3

2015-03-28 1

2015-03-28 2

2015-03-28 3

create table order\_master(orderdate datetime,orderno int primary key(orderdate,orderno))

composite primary key

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

Multiple columns set with single primary key constraint refers to Composite Primary Key.

Composite Primary Key can be created only at columnlevel or Table Level.

Behaviour of Composite Primary Key is that it will allow to store duplicates at 1 column only when corresponding column contains unique data. It will not allow to store null values.

Composite primary key can be set to Min 2 Cols and Max of 16 Columns.

Train\_master

train\_no c\_no s\_no

2655 s1 1

2655 s1 2

2655 s1 3

2655 s2 1

2656 s1 1

create table Train\_master(train\_no int,c\_no char(2),s\_no int primary key(train\_no,c\_no,s\_no))

Foreign key Constraints:

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

\* is used to create the relationship between multiple tables.

\* it accept duplicate values and null values.

\* it refers to the primary key columns of the parent table.

\* After create the relationship parent table records cannot be truncated.

dept1 (Parent table)

deptno(pk)dname loc

10 admin che

20 hr hyd

30 it hyd

40 per hyd

emp1 (child table)

ename empno deptno(fk)salary

aaa 1001 10 25000

bbb 1002 10 3000

syed 1003 20 46000

shekar 1004 null 35000

drop table dept1

go

drop table emp1

go

create table dept1(deptno int primary key,dname varchar(20),loc varchar(20))

insert into dept1 values(10,'admin','che'),(20,'hr','hyd'),(30,'it','hyd'),(40,'per','hyd')

create table emp1(ename varchar(20),empno int ,deptno int foreign key references dept1(deptno),salary int)

foreighn key constraints syntax:

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

fieldname fieldtype foreign key references <parenttablename> (parentcolname)

create table emp2(ename varchar(20),empno int,deptno int foreign key references dept1(deptno),salary int)

insert into emp1 values('aaa',1001,10,2000)

insert emp1 values('bbb',1002,10,3000)

insert emp1 values('ccc',1003,20,40000)

insert emp1 values('ddd',1004,null,3000)

insert emp1 values('eee',1005,50,6500)--invalid

Types of Constriants:

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

1.Column level constriants - New table - create command

2.Table level constraints - Existing table -- alter command

Tabel level constraints:

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

syntax:

alter table <Tablename> add constraint <constraint\_name> <constriant\_type>

select \* from emp1

--how to know howmany constriants are created?

sp\_helpconstraint 'tablename'

sp\_helpconstraint 'emp1'

alter table emp1 add constraint c1\_salary check(salary between 2000 and 40000)

alter table emp1 add constraint c2\_empno primary key(empno)

sp\_help 'emp1'

alter table emp1 alter column empno int not null

sp\_helpconstraint 'emp1'

--how to disable or enable the constraints?

-- Disable all table constraints

ALTER TABLE emp1 NOCHECK CONSTRAINT ALL

-- Enable all table constraints

ALTER TABLE emp1 CHECK CONSTRAINT ALL

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

-- Disable single constraint

ALTER TABLE emp1 NOCHECK CONSTRAINT c1\_salary

-- Enable single constraint

ALTER TABLE emp1 CHECK CONSTRAINT c1\_salary

--how to drop the constraints?

alter table <Tablename> drop constraint <constraint\_name>

alter table emp1 drop constraint FK\_\_emp1\_\_deptno\_\_4AB81AF0

sp\_helpconstraint 'emp1'

Joins

-----

\* Joins are used to reterive the data from the multiple tables or multiple select statements.

Types of join:

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

a.Inner join /Simple join / Natural joins

b.outer join

c.self join

d.cross join or cartesian join

Note:

set operators => adding rows from multiple tables

joins => adding columns from multiples tables.

1.inner join /Simple joins/ natural joins(To reterive the data from the multiple tables.)

a.Equi join (=)

b.Non Equi join (<,>,<=,>=,!=)

2.outer join (To reterive the data from the multiple tables)

a.Left Outer join

b.Right outer join

c.Full outer join

3. Self join (Joining the table itself using table alias name)

4. Cross join / Cartesian join

\* All possible combination values.

\* Without using conditional clause to reterive the data from the multiple tables.

\* Multiply by two tables.

Join Syntax:

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

1.Sql syntax(<,>,<=,>=,!=,=)

2.Ansi syntax( left outer join,right outer join,full outer join,cross join,inner join ....)

--Inner join sql syntax:

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

select table1.\*,table2.\* from table1,table2 where table1.commoncol=table2.commoncol

--inner join ansi syntax:

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

select table1.\*,table2.\* from table1 inner join table2 on table1.commoncol=table2.commoncol

Inner join ansi syntax (3 tables)

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

select table1.\*,table2.\*,table3.\* from table1

inner join table2 on table1.commoncol=table2.commoncol

inner join table3 on table2.commoncol=table3.commoncol

Left outer join syntax:

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

select table1.\*,table2.\* from table1 left outer join table2 on table1.commoncol=table2.commoncol

right outer join syntax:

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

select table1.\*,table2.\* from table1 right outer join table2 on table1.commoncol=table2.commoncol

full outer join syntax:

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

select table1.\*,table2.\* from table1 full outer join table2 on table1.commoncol=table2.commoncol

Cross join syntax:

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

select table1.\*,table2.\* from table1 cross join table2

--display all ename and corresponding deptname

select emp.ename,dept.dname from emp,dept where emp.deptno=dept.deptno

--ansi syntax

select emp.ename,dept.dname from emp inner join dept on emp.deptno=dept.deptno

--display all ename,deptname,location

select emp.ename,dept.DNAME,dept.loc from emp inner join dept on emp.deptno=dept.deptno

--list all ename and corresponding job location

select emp.ename,dept.loc from emp inner join dept on emp.deptno=dept.deptno

hyd

ename empno salary

aaa 1001 10000

bbb 1002 20000

ccc 1003 30000

ddd 1004 40000

eee 1005 50000

che

name empno sal

xxx 1001 2500

yyy 1003 4000

mmm 1005 3500

nnn 1007 3000

zzz 1009 3500

create table hyd(ename varchar(20),empno int,salary int)

create table che(name varchar(20),empno int,sal int)

insert into hyd values('aaa',1001,10000),('bbb',1002,20000),('ccc',1003,30000),

('ddd',1004,40000),('eee',1005,50000)

insert into che values('xxx',1001,2500),

('yyy',1003,4000),('mmm',1005,3500),

('nnn',1007,3000),('zzz',1009,3500)

--Equi join

select che.\*,hyd.\* from che inner join hyd on che.empno=hyd.empno

--Non Equi join

select che.\*,hyd.\* from che inner join hyd on che.empno !=hyd.empno

--left outer join

select che.\*,hyd.\* from che left outer join hyd on che.empno=hyd.empno

--right outer join

select che.\*,hyd.\* from che right outer join hyd on che.empno=hyd.empno

--full outer join

select che.\*,hyd.\* from che full outer join hyd on che.empno=hyd.empno

--cross join

select che.\*,hyd.\* from che cross join hyd

customer

cname amount

aaa 10000

syed 20000

raj 30000

bank

bankname interest

sbi 12

sbh 10

icici 9

select cname,bankname,amount,(amount \* 5\* interest)/100 as si from customer cross join bank

where bankname='sbi'

Self join:(Joining the table itself using table alias name)

--display all ename and mgrname

emp a

ename empno sal mgr deptno

SMITH 7369 800 7902 20

ALLEN 7499 1600 7698 30

WARD 7521 1250 7698 30

JONES 7566 2975 7839 20

MARTIN 7654 1250 7698 30

BLAKE 7698 2850 7839 30

CLARK 7782 2450 7839 10

SCOTT 7788 3000 7566 20

KING 7839 5000 NULL 10

TURNER 7844 1500 7698 30

ADAMS 7876 1100 7788 20

JAMES 7900 950 7698 30

FORD 7902 3000 7566 20

MILLER 7934 1300 7782 10

emp b

ename empno sal mgr deptno

SMITH 7369 800 7902 20

ALLEN 7499 1600 7698 30

WARD 7521 1250 7698 30

JONES 7566 2975 7839 20

MARTIN 7654 1250 7698 30

BLAKE 7698 2850 7839 30

CLARK 7782 2450 7839 10

SCOTT 7788 3000 7566 20

KING 7839 5000 NULL 10

TURNER 7844 1500 7698 30

ADAMS 7876 1100 7788 20

JAMES 7900 950 7698 30

FORD 7902 3000 7566 20

MILLER 7934 1300 7782 10

--self join

select a.ename,b.ename as mgrname from emp a,emp b

where a.mgr=b.empno (13 rows)

ename mgrname

SMITH FORD

ALLEN BLAKE

WARD BLAKE

JONES KING

MARTIN BLAKE

BLAKE KING

CLARK KING

SCOTT JONES

TURNER BLAKE

ADAMS SCOTT

JAMES BLAKE

FORD JONES

MILLER CLARK

--left outer join

--display ename and corresponding mgrname

select a.ename,b.ename as mgrname from emp a

left outer join emp b on a.mgr=b.empno (14 rows)

ename mgrname

SMITH FORD

ALLEN BLAKE

WARD BLAKE

JONES KING

MARTIN BLAKE

BLAKE KING

CLARK KING

SCOTT JONES

KING NULL

TURNER BLAKE

ADAMS SCOTT

JAMES BLAKE

FORD JONES

MILLER CLARK

customer

cname amount

aaa 10000

bbb 12000

ccc 15000

bank

bankname interest

sbi 10

sbh 9

icici 8

create table customer(cname varchar(20),amount int)

insert into customer values('aaa',10000),('bbb',12000),('ccc',15000)

create table bank(bankname varchar(20),interest int)

insert into bank values('sbi',10),('sbh',9),('icici',8)

select cname,bankname,amount,(Amount \* 5 \*interest)/100

as si from customer cross join bank

Joins

-----

\* Joins are used to reterive the data from the multiple tables or multiple select statements.

Types of join:

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

a.Inner join /Simple join / Natural joins

b.outer join

c.self join

d.cross join or cartesian join

Note:

set operators => adding rows from multiple tables

joins => adding columns from multiples tables.

1.inner join /Simple joins/ natural joins(To reterive the data from the multiple tables.)

a.Equi join (=)

b.Non Equi join (<,>,<=,>=,!=)

2.outer join (To reterive the data from the multiple tables)

a.Left Outer join

b.Right outer join

c.Full outer join

3. Self join (Joining the table itself using table alias name)

4. Cross join / Cartesian join

\* All possible combination values.

\* Without using conditional clause to reterive the data from the multiple tables.

\* Multiply by two tables.

Join Syntax:

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

1.Sql syntax(<,>,<=,>=,!=,=)

2.Ansi syntax( left outer join,right outer join,full outer join,cross join,inner join ....)

--Inner join sql syntax:

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

select table1.\*,table2.\* from table1,table2 where table1.commoncol=table2.commoncol

--inner join ansi syntax:

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

select table1.\*,table2.\* from table1 inner join table2 on table1.commoncol=table2.commoncol

Inner join ansi syntax (3 tables)

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

select table1.\*,table2.\*,table3.\* from table1

inner join table2 on table1.commoncol=table2.commoncol

inner join table3 on table2.commoncol=table3.commoncol

Left outer join syntax:

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

select table1.\*,table2.\* from table1 left outer join table2 on table1.commoncol=table2.commoncol

right outer join syntax:

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

select table1.\*,table2.\* from table1 right outer join table2 on table1.commoncol=table2.commoncol

full outer join syntax:

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

select table1.\*,table2.\* from table1 full outer join table2 on table1.commoncol=table2.commoncol

Cross join syntax:

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

select table1.\*,table2.\* from table1 cross join table2

--display all ename and corresponding deptname

select emp.ename,dept.dname from emp,dept where emp.deptno=dept.deptno

--ansi syntax

select emp.ename,dept.dname from emp inner join dept on emp.deptno=dept.deptno

--display all ename,deptname,location

select emp.ename,dept.DNAME,dept.loc from emp inner join dept on emp.deptno=dept.deptno

--list all ename and corresponding job location

select emp.ename,dept.loc from emp inner join dept on emp.deptno=dept.deptno

hyd

ename empno salary

aaa 1001 10000

bbb 1002 20000

ccc 1003 30000

ddd 1004 40000

eee 1005 50000

che

name empno sal

xxx 1001 2500

yyy 1003 4000

mmm 1005 3500

nnn 1007 3000

zzz 1009 3500

create table hyd(ename varchar(20),empno int,salary int)

create table che(name varchar(20),empno int,sal int)

insert into hyd values('aaa',1001,10000),('bbb',1002,20000),('ccc',1003,30000),

('ddd',1004,40000),('eee',1005,50000)

insert into che values('xxx',1001,2500),

('yyy',1003,4000),('mmm',1005,3500),

('nnn',1007,3000),('zzz',1009,3500)

--Equi join

select che.\*,hyd.\* from che inner join hyd on che.empno=hyd.empno

--Non Equi join

select che.\*,hyd.\* from che inner join hyd on che.empno !=hyd.empno

--left outer join

select che.\*,hyd.\* from che left outer join hyd on che.empno=hyd.empno

--right outer join

select che.\*,hyd.\* from che right outer join hyd on che.empno=hyd.empno

--full outer join

select che.\*,hyd.\* from che full outer join hyd on che.empno=hyd.empno

--cross join

select che.\*,hyd.\* from che cross join hyd

customer

cname amount

aaa 10000

syed 20000

raj 30000

bank

bankname interest

sbi 12

sbh 10

icici 9

select cname,bankname,amount,(amount \* 5\* interest)/100 as si from customer cross join bank

where bankname='sbi'

Self join:(Joining the table itself using table alias name)

--display all ename and mgrname

emp a

ename empno sal mgr deptno

SMITH 7369 800 7902 20

ALLEN 7499 1600 7698 30

WARD 7521 1250 7698 30

JONES 7566 2975 7839 20

MARTIN 7654 1250 7698 30

BLAKE 7698 2850 7839 30

CLARK 7782 2450 7839 10

SCOTT 7788 3000 7566 20

KING 7839 5000 NULL 10

TURNER 7844 1500 7698 30

ADAMS 7876 1100 7788 20

JAMES 7900 950 7698 30

FORD 7902 3000 7566 20

MILLER 7934 1300 7782 10

emp b

ename empno sal mgr deptno

SMITH 7369 800 7902 20

ALLEN 7499 1600 7698 30

WARD 7521 1250 7698 30

JONES 7566 2975 7839 20

MARTIN 7654 1250 7698 30

BLAKE 7698 2850 7839 30

CLARK 7782 2450 7839 10

SCOTT 7788 3000 7566 20

KING 7839 5000 NULL 10

TURNER 7844 1500 7698 30

ADAMS 7876 1100 7788 20

JAMES 7900 950 7698 30

FORD 7902 3000 7566 20

MILLER 7934 1300 7782 10

--self join

select a.ename,b.ename as mgrname from emp a,emp b

where a.mgr=b.empno (13 rows)

ename mgrname

SMITH FORD

ALLEN BLAKE

WARD BLAKE

JONES KING

MARTIN BLAKE

BLAKE KING

CLARK KING

SCOTT JONES

TURNER BLAKE

ADAMS SCOTT

JAMES BLAKE

FORD JONES

MILLER CLARK

--left outer join

--display ename and corresponding mgrname

select a.ename,b.ename as mgrname from emp a

left outer join emp b on a.mgr=b.empno (14 rows)

ename mgrname

SMITH FORD

ALLEN BLAKE

WARD BLAKE

JONES KING

MARTIN BLAKE

BLAKE KING

CLARK KING

SCOTT JONES

KING NULL

TURNER BLAKE

ADAMS SCOTT

JAMES BLAKE

FORD JONES

MILLER CLARK

customer

cname amount

aaa 10000

bbb 12000

ccc 15000

bank

bankname interest

sbi 10

sbh 9

icici 8

create table customer(cname varchar(20),amount int)

insert into customer values('aaa',10000),('bbb',12000),('ccc',15000)

create table bank(bankname varchar(20),interest int)

insert into bank values('sbi',10),('sbh',9),('icici',8)

select cname,bankname,amount,(Amount \* 5 \*interest)/100

as si from customer cross join bank

Mock test url path

http://sqlschool.com/RTCS\_SQL2.html

SUB-QUERIES:

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

A Sub Query is also a query, which is defined under a main query.

First Query is also called as outer query or main query and second query is also called as inner query or subquery.

It always sends values to its nearest main query.

Server first sub query executed based on the sub query value main query will get executed.

If a sub query send single value to its nearest main query then that sub query is called Single-Valued-Sub query.

If a sub query sends multiple values to its nearest main query then that sub query is called Multi-Valued-Sub query.

Note: If a sub query sends multiple values to its nearest main query then we have to use IN or NOT IN operator between Main query and Sub query.

Single valued Subquery:

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

SELECT \* FROM TABLENAME WHERE COLUMNNAME =

(SELECT STATEMENT WHERE CONDITION)

Multi valued subquery:

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

select \* from tablename where columname in

(select statement where condition)

Types of Subquery:

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

1.Nested Subquery

2.Correlated Subquery

--display all emp details whose earning salary above avg.salary

select \* from emp where sal>=(select avg(sal) from emp)

--display all emp details whose earning first highest salary

select \* from emp where sal=(select max(sal) from emp)

--display all emp details whose earning Second highest salary

select \* from emp where sal=(select max(sal) from emp where sal<(select max(sal) from emp))

--display all emp details whose earning 3rd highest salary

select \* from emp where sal=(select max(sal) from emp where sal<(select max(sal) from emp

where sal<(select max(sal) from emp)))

--display all emp details whose earning nth highest salary where n=10

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank=10

--display all emp details whose earning first 3 highest salary

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank<=3

--display all emp details whose earning whose earning salary between 5 to 10th highest salary

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank between 5 and 10

--5 to 10 and 12 to 14

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank between 5 and 10 or rank between 12 and 14

--display all emp details whose earning least salary

select \* from emp where sal=(select min(sal) from emp)

--display all emp details whose earning salary above avg.salary of deptno 10

select \* from emp where sal >=(select avg(sal) from emp where deptno=10)

--display all emp details whose earning salary above avg.salary of deptno 20

select \* from emp where sal >=(select avg(sal) from emp where deptno=20)

--display all emp details whose earning salary above avg.salary of deptno 30

select \* from emp where sal >=(select avg(sal) from emp where deptno=30)

--display all emp details who are working along with smith dept

select \* from emp where deptno=(select deptno from emp where ename='Smith')

--To modify the smith salary to the highest salary and transer to ford job.

update emp set sal=(Select max(sal) from emp),job=(select job from emp

where ename='Ford') where ename='Smith'

--To modify the smith salary to the 2nd highest salary and transer to ford job.

update emp set sal=(select max(sal) from emp where sal <(Select max(sal) from emp)),

job=(select job from emp where ename='ford') where ename='smith'

ename eno salary

aaa 101 10000

bbb 102 20000

ccc 103 30000

ddd 104 40000

eee 105 50000

anil 101 35000

guna 105 50000

--how to select duplicate records

select \* from emp2 where %%physloc%% not in(select min(%%physloc%%) as min from emp2 group by eno)

--how to delete duplicate records?

delete from emp2 where %%physloc%% not in(select min(%%physloc%%) as min from emp2 group by eno)

SELECT \* FROM EMP2

-- WAQ To display employee details, who are working under SMITH dept.

select \* from emp where deptno=(select deptno from emp where ename='smith')

--how to select duplicate records?

select \* from emp1 where %%physloc%% not in(select min(%%physloc%%) as min from emp1 group by empno)

--How to delete the duplicate records?

delete from emp1 where %%physloc%% not in (select min(%%physloc%%) as min from emp1 group by empno)

\*SUB-Queries with Update command:

\* Update the employee salary to max-salary whose eno=7369

update emp set sal=(Select max(sal) from emp) where empno=7369

\* UPDATE THE EMPLOYEE SALARY TO MAX-SALARY AND CHANGE JOB 'FORD' WHERE EMPNO=7566

update emp set sal=(select max(sal) from emp),job=(select job from emp where ename='Ford')

where empno=7566

--Display ename,job,sal of all those employees whose row positions are at EVEN positions in ascending order of salary?

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn%2=0

--odd series rows

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn%2=1

--4th series rows

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn%4=0

--first records

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn=1

--5th to 10th records

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn between 5 and 10

--last records

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn=(Select count(\*) from emp)

--how to select duplicate records?

select empno from emp1 group by empno having count(\*)>1

--how to delete the duplicate records? (FAQ)

delete from emp1 where %%physloc%% not in(select min(%%physloc%%) from emp1 group by empno)

--how to display reverse order

select \*,%%physloc%% from emp2 order by %%physloc%% desc

Types of Subquery:

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

1.Nested Subquery -- Table level execution

2.Correlated Subquery - row level execution

select ename,empno,sal,deptno from emp e

where sal >=(select avg(sal) from emp where deptno=e.deptno)

CORRELATED SUB QUERIES:

These queries provide different execution to nested query, where it first executes Outer query and then executes inner query.

Nested query is called uni-directional query whereas Corelated Sub Query is called bi-directional query.

Corelated Sub Query will always gets executed in the following way;

"First outer query gets executed, extracts 1 row at a time(candidate row) and that row is given to inner query for processing, inner query will provide its output to outer query and based on the condition outer query will display the extracted record".

--Example:

---------

--display all emp details whose earning salary above avg.salary of respective dept

select ename,empno,sal,deptno,mgr from emp e where sal>=

(select avg(sal) from emp where deptno=e.deptno)

--display max salary for each dept and corresponding emp details

select \* from emp e where sal=(select max(sal) from emp where deptno=e.deptno)

--display all emp details whose earning salary nth highest salary where n=10

select \* from emp e where 10=(select count(distinct (sal)) from emp where sal>=e.sal)

XML integration:

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

\* to convert the table into xml format

\* to convert the xml into table format

\* use xml datatype

1.How to convert table into xml format?

select \* from emp for xml auto

select \* from emp for xml raw

select \* from emp for xml path

2.how to convert xml into table?

use :ssis package

3.how to create xml datatype?

create table empx(eno int,addr xml)

insert into empx values(1001,'<city>hyd</city><state>TG</state>')

insert into empx values(1002,'<city>chennai</city><state>TN</state>')

select \* from empx

Backup : is nothing but a taking duplicate copy of the database.

restore : is nothing but a reterive data from backup devices into database.

syntax:

backup database <databasename> to disk='physical path' --online

restore database <databasename> from disk='physical path',replace,recovery --offline

replace -> overwriting data to the database

recovery -> after restore user can and read write

practial step1:

step1 : backup database Geetha to disk='c:\Geetha.bak'

step2 : delete some tables or delete some objects.

step3 : close sqlserver connection

step4 : restore database Geetha from disk='c:\Geetha.bak' with replace,recovery

step5 : Verify the tables.

Database objects:

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

table

view

synonym

index

stored procedure

stored function

stored trigger or database trigger

Sequence (from sqlserver 2012)

Mock test path:

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

http://sqlschool.com/RTCS\_SQL2.html

SUB-QUERIES:

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

A Sub Query is also a query, which is defined under a main query.

First Query is also called as outer query or main query and second query is also called as inner query or subquery.

It always sends values to its nearest main query.

Server first sub query executed based on the sub query value main query will get executed.

If a sub query send single value to its nearest main query then that sub query is called Single-Valued-Sub query.

If a sub query sends multiple values to its nearest main query then that sub query is called Multi-Valued-Sub query.

Note: If a sub query sends multiple values to its nearest main query then we have to use IN or NOT IN operator between Main query and Sub query.

Single valued Subquery:

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

SELECT \* FROM TABLENAME WHERE COLUMNNAME =

(SELECT STATEMENT WHERE CONDITION)

Multi valued subquery:

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

select \* from tablename where columname in

(select statement where condition)

Types of Subquery:

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

1.Nested Subquery

2.Correlated Subquery

--display all emp details whose earning salary above avg.salary

select \* from emp where sal>=(select avg(sal) from emp)

--display all emp details whose earning first highest salary

select \* from emp where sal=(select max(sal) from emp)

--display all emp details whose earning Second highest salary

select \* from emp where sal=(select max(sal) from emp where sal<(select max(sal) from emp))

--display all emp details whose earning 3rd highest salary

select \* from emp where sal=(select max(sal) from emp where sal<(select max(sal) from emp

where sal<(select max(sal) from emp)))

--display all emp details whose earning nth highest salary where n=10

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank=10

--display all emp details whose earning first 3 highest salary

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank<=3

--display all emp details whose earning whose earning salary between 5 to 10th highest salary

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank between 5 and 10

--5 to 10 and 12 to 14

select \* from (select ename,empno,sal,deptno,mgr,comm,job,dense\_rank() over(order by sal desc) as rank from emp) e

where rank between 5 and 10 or rank between 12 and 14

--display all emp details whose earning least salary

select \* from emp where sal=(select min(sal) from emp)

--display all emp details whose earning salary above avg.salary of deptno 10

select \* from emp where sal >=(select avg(sal) from emp where deptno=10)

--display all emp details whose earning salary above avg.salary of deptno 20

select \* from emp where sal >=(select avg(sal) from emp where deptno=20)

--display all emp details whose earning salary above avg.salary of deptno 30

select \* from emp where sal >=(select avg(sal) from emp where deptno=30)

--display all emp details who are working along with smith dept

select \* from emp where deptno=(select deptno from emp where ename='Smith')

--To modify the smith salary to the highest salary and transer to ford job.

update emp set sal=(Select max(sal) from emp),job=(select job from emp

where ename='Ford') where ename='Smith'

--To modify the smith salary to the 2nd highest salary and transer to ford job.

update emp set sal=(select max(sal) from emp where sal <(Select max(sal) from emp)),

job=(select job from emp where ename='ford') where ename='smith'

ename eno salary

aaa 101 10000

bbb 102 20000

ccc 103 30000

ddd 104 40000

eee 105 50000

anil 101 35000

guna 105 50000

--how to select duplicate records

select \* from emp2 where %%physloc%% not in(select min(%%physloc%%) as min from emp2 group by eno)

--how to delete duplicate records?

delete from emp2 where %%physloc%% not in(select min(%%physloc%%) as min from emp2 group by eno)

SELECT \* FROM EMP2

-- WAQ To display employee details, who are working under SMITH dept.

select \* from emp where deptno=(select deptno from emp where ename='smith')

--how to select duplicate records?

select \* from emp1 where %%physloc%% not in(select min(%%physloc%%) as min from emp1 group by empno)

--How to delete the duplicate records?

delete from emp1 where %%physloc%% not in (select min(%%physloc%%) as min from emp1 group by empno)

\*SUB-Queries with Update command:

\* Update the employee salary to max-salary whose eno=7369

update emp set sal=(Select max(sal) from emp) where empno=7369

\* UPDATE THE EMPLOYEE SALARY TO MAX-SALARY AND CHANGE JOB 'FORD' WHERE EMPNO=7566

update emp set sal=(select max(sal) from emp),job=(select job from emp where ename='Ford')

where empno=7566

--Display ename,job,sal of all those employees whose row positions are at EVEN positions in ascending order of salary?

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn%2=0

--odd series rows

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn%2=1

--4th series rows

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn%4=0

--first records

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn=1

--5th to 10th records

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn between 5 and 10

--last records

select \* from (select \*,row\_number() over(order by sal desc) as rn from emp) e

where rn=(Select count(\*) from emp)

--how to select duplicate records?

select empno from emp1 group by empno having count(\*)>1

--how to delete the duplicate records? (FAQ)

delete from emp1 where %%physloc%% not in(select min(%%physloc%%) from emp1 group by empno)

--how to display reverse order

select \*,%%physloc%% from emp2 order by %%physloc%% desc

Types of Subquery:

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

1.Nested Subquery -- Table level execution

2.Correlated Subquery - row level execution

select ename,empno,sal,deptno from emp e

where sal >=(select avg(sal) from emp where deptno=e.deptno)

CORRELATED SUB QUERIES:

These queries provide different execution to nested query, where it first executes Outer query and then executes inner query.

Nested query is called uni-directional query whereas Corelated Sub Query is called bi-directional query.

Corelated Sub Query will always gets executed in the following way;

"First outer query gets executed, extracts 1 row at a time(candidate row) and that row is given to inner query for processing, inner query will provide its output to outer query and based on the condition outer query will display the extracted record".

--Example:

---------

--display all emp details whose earning salary above avg.salary of respective dept

select ename,empno,sal,deptno,mgr from emp e where sal>=

(select avg(sal) from emp where deptno=e.deptno)

--display max salary for each dept and corresponding emp details

select \* from emp e where sal=(select max(sal) from emp where deptno=e.deptno)

--display all emp details whose earning salary nth highest salary where n=10

select \* from emp e where 10=(select count(distinct (sal)) from emp where sal>=e.sal)

XML integration:

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

\* to convert the table into xml format

\* to convert the xml into table format

\* use xml datatype

1.How to convert table into xml format?

select \* from emp for xml auto

select \* from emp for xml raw

select \* from emp for xml path

2.how to convert xml into table?

use :ssis package

3.how to create xml datatype?

create table empx(eno int,addr xml)

insert into empx values(1001,'<city>hyd</city><state>TG</state>')

insert into empx values(1002,'<city>chennai</city><state>TN</state>')

select \* from empx

Backup : is nothing but a taking duplicate copy of the database.

restore : is nothing but a reterive data from backup devices into database.

syntax:

backup database <databasename> to disk='physical path' --online

restore database <databasename> from disk='physical path',replace,recovery --offline

replace -> overwriting data to the database

recovery -> after restore user can and read write

practial step1:

step1 : backup database Geetha to disk='c:\Geetha.bak'

step2 : delete some tables or delete some objects.

step3 : close sqlserver connection

step4 : restore database Geetha from disk='c:\Geetha.bak' with replace,recovery

step5 : Verify the tables.

Database objects:

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

1.table

2.view

3.synonym

4.index

5.stored procedure

6.stored function

7.stored trigger or database trigger

8.Sequence (From sqlserver 2012)

VIEWS:

----------

\* A View is nothing but an image table or virtual table,which is created for a base table.

\* A view can be created by taking all values from the base table or by taking only selected values from base table.

There are two types views available in SQL Server.

1. Simple Views(Updateable view)

2. Complex Views (Non -Updateable View)

Note: If we perform any modifications in base table, then those modifications automatically effected in view and vice-versa.

\* View does not contain data.

\* View does not allocate physical memory.

\* It supports DML and DQL (insert,update,delete,select)

1.Simple Views: Creating View by taking only one single base table.

Syntax:

CREATE VIEW VIEWNAME [WITH ENCRYPTION]

[WITH SCHEMABINDING]

AS SELECT \* FROM TABLENAME [WHERE CONDITION]

[WITH CHECK OPTION]

DBO - DATABASE OWNER

create view v1 as select \* from emp

select \* from v1

update v1 set ename='Anil' where ename='Adams'

select \* from emp

create view v2 as select ename,empno,sal from emp

select \* from v2

create view v3 as select \* from emp where deptno=10

select \* from v3

create view v4 as select emp.ename,dept.dname from emp inner join dept on emp.deptno=dept.deptno

select \* from v4

create view v5 as select \* from v1

--how to know howmany views are created?

select \* from sys.views

--how to view the view code?

sp\_helptext 'viewname'

sp\_helptext 'v1'

sp\_helptext 'v2'

--how to hide the view code?

alter view v1 with encryption as select \* from emp

sp\_helptext 'v1'

E.g.:

CREATE VIEW V2 AS SELECT \* FROM EMP WHERE DEPTNO=10

INSERT INTO V2 VALUES (66,’BABBU’, 25000,10)

The above insert statement inserts the values into base table EMP as well as into view

V2.

INSERT INTO V2 VALUES (77,’AMAR’, 15000, 20)

The above insert statement inserts the values into only base table EMP but not into view

V2 because according to the definition of V2 user supplied values are invalid values.

It means invalid values are inserting into base table EMP.

To stop this kind of operations we have to create the view with ‘WITH CHECK OPTION’.

E.g.:

CREATE VIEW V3 AS SELECT \* FROM EMP WHERE DEPTNO=10

WITH CHECK OPTION

INSERT INTO V3 VALUES (88,’TEJA’, 25000,20)

The above insert statement cannot inserts the values into base table EMP as well as into view V3.

sp\_helptext 'v3'

select \* from v3

select \* from emp

insert into v3(ename,empno,sal,deptno) values('guna',1002,2500,20)

alter view v3 as select \* from emp where deptno=10 with check option

Schema Binding :

a. when view created by schemabinding then view becomes dependent object on base table.

b. The base table cannot be dropped without dropping view.

c. When schemabinding is used then tablename prefixed by username/schemaname.

ex:

create view v8 with schemabinding

as select empno,ename,sal,job from dbo.emp;

ex: drop table emp

Invalid , because view is dependent object so we first drop the view after that table.

--how to drop views?

drop view <viewname>

drop view v1

complex view: (Read only view or non updateable view)

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

By default a complex view is not updatable view (i.e. read only view).

A view is said to be a complex views

consists any one of the following

a.joins

b.mathematical expressions

c.group by clause

d.group functions

e.distinct operator.

create view v11 as select ename,empno,sal,sal\*12 as ctc from emp

select \* from v11

update v11 set ctc=ctc+100 --INVALID (COMPLEX VIEW)

Advantages of view:

1. To hide the complexity of the underlying database schema, or customize the data and schema for a set of users.

2. Restrict access to specific rows in a table.

3. Restrict access to specific columns in a table.

4. Join columns from multiple tables and present them as though they are part of a single table

5 Present aggregate information (such as the results of the COUNT function)

drop view viewname;

Synonyms:

--------

SYNONYM OBJECT:

\* SYNONYM OBJECT IS USED TO CREATE THE permanent ALIAS NAME for single base table OR VIEW.

\* IT IS USED TO HIDE THE ORIGINAL NAME OF AN OBJECT.

\* IT IS USED TO CREATE THE SECURITY OF THE TABLE.

\* it supports DML AND DQL OPERATIONS.(select,insert,update,delete)

Note if any changes in the synonym automatically reflected into base table and vice versa.

organization\_master om

SYNTAX:

CREATE SYNONYM <SYNONYMNAME> FOR <TABLENAME>

--example1

create synonym fr for fuzzyreference

select \* from fuzzyreference

insert into fr values(7,'sss')

SELECT \* FROM EMP

CREATE SYNONYM S1 FOR EMP

SELECT \* FROM S1

DELETE FROM S1 WHERE ENO=101

DELETE FROM S1 WHERE ENO=101

DROP SYNONYM S1

SELECT \* FROM S1

select \* from sys.synonyms

where base\_object\_name='[emp]'

create synonym fm for fuzzylookupmatchindex

create synonym fm1 for fuzzylookupmatchindex

select \* from fm

select \* from fm1

select \* from sys.synonyms where base\_object\_name='[fuzzylookupmatchindex]'

--how to know howmany synonyms are created?

select \* from sys.synonyms

select name,base\_object\_name from sys.synonyms

--how to drop the synonym

drop synonym fr

Index:

-------

select \* from emp

(scanning)

In Sqlserver we have two types of scanning:

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

1.Tablescan - Sequential order -linear searching --small amount of data --by default

2.Indexscan - Random order -- binary searching --large amount of data --user manually creating

What is an Index?

\* Index is used to search for required rows quickly.

\* Index occupies extra space. Index is stored separately from table.

\* To speed up searching for a particular value thereby improving performance of query.

\* To enforce uniqueness

How does Index work?

\* Indexing is a way of sorting a number of records on multiple fields.

\* Creating an index on a field in a table creates another data structure(binary Tree) which holds the field value, and a pointer to the record it relates to.

\* This index structure is then sorted, allowing Binary Searches to be performed on it.

Types of index

1.clustered index

2.Non clustered index (by default)

3.Unique index

syntax:

create clustered index <indexname> on <tablename>(colname)

create nonclustered index <indexname> on <tablename>(colname)

create unique index <indexname> on <Tablename>(colname)

SYNTAX:

CREATE [UNIQUE]/[CLUSTERED]/[NONCLUSTERED] INDEX index\_name

ON TABLE\_NAME(COL1[,COL2,COL3,.......])

When index is created on multiple columns, it is called Composite Index.

I) Clustered Index:

\* Clustered index will alter the physical representation of rows in a table.

\* A table can have only 1 clustered index.

\* It will always arrange the data in sorted order.

\* Data Pages and Index pages will be stored at one level.

\* This index should be created for a column where more than 1 search value is available.

eno

110

101

105

102

130

102

103

clustered index

dat index

101-1

102-2

102-3

103-4

105-5

110-6

130-7

II) Non Clustered Index:

This index will not alter the physical representation of rows in a table.

A table can have 249 Non clustered indexes.(sqlserver 2005).

A table can have 999 Non Clustered indexes(sqlserver 2008).

Data is not arranged in order.

It is a default index created.

Data pages and Index pages are stored at different levels.

data index

110 - 5

101 - 2

102 - 3

100 - 1

105 - 4

Example:

CREATE NONCLUSTERED INDEX i2 ON EMP(ENAME)

CREATE INDEX i3 ON EMP(JOB)

III) UNIQUE INDEX:

This index can be created only on those columns which contains unique data.

This index is automatically created

when unique constraint or primary key is created on a column.

Example:

CREATE UNIQUE INDEX i4 ON EMP(EMPNO)

NOTE:

SQL Server will automatically create an index for 2 constraints,Primary key and unique.

Rebuild Index task :

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

\* The Rebuild Index task rebuilds indexes in SQL Server database tables and views.

\*By using the Rebuild Index task, a package can rebuild indexes in a single database or multiple databases.

Reorganize Index task:

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

\* It reorganizes indexes in SQL Server database tables and views.

\*By using the Reorganize Index task, a package can reorganize indexes in a single database or multiple databases.

\* If the task reorganizes only the indexes in a single database, you can choose the views or the tables whose indexes the task reorganizes.

TO REBUILD AN INDEX:

SYNTAX:

ALTER INDEX index\_name ON object\_name REBUILD/REORAGANIZE

EXAMPLE:

ALTER INDEX I1 ON emp10 REBUILD

select \* from emp

create clustered index i1 on emp(Deptno)

create nonclustered index i2 on emp(sal)

create unique index i3 on emp(empno)

--HOW TO know howmany index are created?

sp\_helpindex 'tablename'

sp\_helpindex 'emp'

--how to drop index?

drop index tablename.indexname

drop index emp.i1

--how to create a new table from existing table?

select \* into <newtablename> from <oldtablename> where <condition>

select \* into empnew1 from emp

select \* from empnew1

select \* into empnew2 from emp where deptno=10

select \* from empnew2

select ename,empno,sal into empnew3 from emp

select \* from empnew3

insert into empnew4 select \* from emp

--Xml integration: (Extensible markup language)

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

\* Table into xml format

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

select \* from emp for xml auto

select \* from emp for xml raw

select \* from emp for xml path

Tsql :

-------

Transact Strucutred Query Language:

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

sqlserver

---------

1.SQL (Non procedural language)

2.TSQL(Procedural language)

(TRANSACT SQL)

T-SQL deals with set of statements that are grouped in the form of a block and submitted to server at once for execution.

T-SQL (Transact-SQL) is a set of programming Language including transaction control, exception and error handling, row processing, control structure and declared variables.

Microsoft's SQL Server and Sybase's SQL server support T-SQL statements.

select \* from emp

Difference between SQL and T-SQL

"SQL is non-procedural language

T-SQL is a procedural language

SQL was developed by IBM.

T-SQL was developed by Microsoft.

T-SQL supports to submit a block to

Server for execution in 2 ways;

1. Anonymous Block : It supports to store a block in .sql file -Nameless block - Everytime compilation and Every Time Executed - operating system - no security

2. Stored Block : It stores the block in database server. - Named Block - only one time compilation and every time executed. - more security

a) Stored Procedures

b) Stored Functions

c) Stored Triggers/Database Triggers

Variable is nothing but a name of the memory location.

Types of variable:

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

1.Global Variable / Predefined variable / System Defined variable - starts with @@ symbol

2.Local variable / User defined variable - starts with @ symbol

Global Variables:

These variables are provided by software.

These variables cant be used to initialize the value manually.

These variables will be stored with the data automatically based on the operations performed by SQL statements.

These variables are preceeded by @@.

1. @@SERVERNAME : Returns name of the server

SELECT @@SERVERNAME

2. @@SERVICENAME : Returns name of the service(instance name)

SELECT @@SERVICENAME

3. @@VERSION : Returns version of SQL Server

SELECT @@VERSION

4. @@ROWCOUNT : Returns number of rows affected by last executed SQL statement.

SELECT @@ROWCOUNT

5. @@ERROR : Returns 0 if there is no errors in the

last executed SQL statement OR returns Error number.

SELECT @@ERROR

SELECT @@IDENTITY

SELECT @@FETCH\_STATUS --cursor progam

Local Variables:

These variables are created by a user.

These variables are provided with memory for temporary, hence they are called as Buffer Variables.

These variables should be declared before they are used.

These variables will allow to store only 1 value.

These variables will be preceeded by @.

How to declare the variable?

SYNTAX:

DECLARE @VAR1 DATATYPE1,@VAR2 DATATYPE2,@VAR3 DATATYPE3,.......

EX:

DECLARE @A INT,@B VARCHAR(20),@C DATETIME

HOW TO ASSING A VALUE?

SYNTAX:

SET @VAR=VALUE

EX:

SET @A=100

SET @B=200

SET @C=@A+@B

set @d=getdate()

or

SELECT @A=100,@B=200,@C=@A+@B

HOW TO DISPLY THE OUTPUT ?

PRINT @VAR/VALUE

--how to execute TSQL program?

F5 - execute

F11 - step by step (errorchecking)

--Write a TSQL program to calculate addition of two numbers

Declare @a int,@b int,@c int

set @a=1000

set @b=200

set @c=@a+@b

print 'add two values='+cast(@c as varchar)

--tsql program to calculate all arithmetical operators:

I method

----------

declare @a int,@b int,@c int,@d int,@e int,@f int,@g int

set @a=1000

set @b=200

set @c=@a+@b

set @d=@a-@b

set @e=@a\*@b

set @f=@a/@b

set @g=@a%@b

print 'addition='+cast(@c as varchar)

print 'subtract='+cast(@d as varchar)

print 'multiply='+cast(@e as varchar)

print 'quotient='+cast(@f as varchar)

print 'remainder='+cast(@g as varchar)

--II method

--tsql program to calculate all arithmetical operators:

declare @a int,@b int,@c int,@d int,@e int,@f int,@g int

select @a=100,@b=20,@c=@a+@b,@d=@a-@b,@e=@a\*@b,@f=@a/@b,@g=@a%@b

print 'addition='+cast(@c as varchar)

print 'subtract='+cast(@d as varchar)

print 'multiply='+cast(@e as varchar)

print 'quotient='+cast(@f as varchar)

print 'remainder='+cast(@g as varchar)

III method

--tsql program to calculate all arithmetical operators

declare @a int,@b int,@c int,@d int,@e int,@f int,@g int

select @a=1000,@b=20,@c=@a+@b,@d=@a-@b,@e=@a\*@b,@f=@a/@b,@g=@a%@b

print 'addition='+cast(@c as varchar) +char(13) + 'subtract='+cast(@d as varchar)+

char(13)+'multiply='+cast(@e as varchar) +char(13)+ 'quotient='+cast(@f as varchar)

+char(13)+ 'remainder='+cast(@g as varchar)

CONTROL STRUCTURE:

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

\* Control structrure refers to the order in which the various instruction in a programs are to be executed.

\* The other name for control sturcture "flow of control".

control structrue classified into three types:

1.SEQUENTIAL CONTROL STRUCTURE - ONE BY ONE

2.SELECTIVE CONTROL STURCUTRE - TO ALLOWS TO CREATE THE CONDITION

3.REPETATIVE CONTROL STRUCUTRE - TO ALLOWS TO CREATE THE LOOP

SELECTIVE CONTROL STRUCTURE:

IF STATEMENT

IF ELSE STATEMENT

IF ELSE IF STATEMENT

NESTED IF STATEMENT

case statement

IF STATEMENT SYNTAX:

1.IF CONDTION

BEGIN

STATEMENT1

STATEMENT2

END

If...else statement

2.IF CONDITION

BEGIN

STATEMENT1

END

ELSE

BEGIN

STATEMENT2

END

or

IF CONDITION

STATEMENT1

ELSE

STATEMENT2

IF....ELSE ...IF STATEMENT

3.IF CONDITION1

BEGIN

STATEMENT1

END

ELSE IF CONDITION2

BEGIN

STATEMENT2

END

ELSE

BEGIN

STATEMENT3

END

OR

IF CONDITION1

STATEMENT1

ELSE IF CONDITION2

STATEMENT2

ELSE

STATEMENT3

4.CASE VARIABLE

WHEN 1 THEN 'ST1'

WHEN 2 THEN 'ST2'

WHEN 3 THEN 'ST3'

ELSE

STX

END

--WRITE A TSQL PROGRAM TO FIND THE GIVEN MARK IS PASS OR FAIL

DECLARE @MARK INT

SET @MARK=10

IF @MARK>=40

PRINT 'THE GIVEN MARK IS PASS='+CAST(@MARK AS VARCHAR)

ELSE

PRINT 'THE GIVEN MARK IS FAIL='+CAST(@MARK AS VARCHAR)

--TSQL PROGRAM TO FIND THE GREATEST OF TWO NUMBERS

DECLARE @A INT,@B INT

SET @A=1000

SET @B=2000

IF @A>@B

PRINT 'A IS GREATER='+CAST(@A AS VARCHAR)

ELSE

PRINT 'B IS GREATER='+CAST(@B AS VARCHAR)

--tsql program to find the greatest of 3 numbers

DECLARE @A INT,@B INT,@C INT

SET @A=1000

SET @B=20000

SET @C=300

IF @A>@B AND @A>@C

PRINT 'A IS GREATER='+CAST(@A AS VARCHAR)

ELSE IF @B>@C

PRINT 'B IS GREATER='+CAST(@B AS VARCHAR)

ELSE

PRINT 'C IS GREATER='+CAST(@C AS VARCHAR)

--Tsql program to find the given year is leap year or not

declare @y int

set @y=year(getdate())

if @y % 4 =0

print 'Given year is leap year='+cast(@y as varchar)

else

print 'Given year is not leap year='+cast(@y as varchar)

--tsql program to convert the number into day of week

1 - SUNDAY

2 - MONDAY

3 - TUESDAY

4 - WEDNESDAY

5 - THURSDAY

6- FRIDAY

7 - SAT

>7 - INVALID DAY

declare @day int,@result varchar(20)

set @day=1

select @result=case @day

when 1 then 'Sunday'

when 2 then 'Monday'

when 3 then 'Tuesday'

when 4 then 'Wednesday'

when 5 then 'Thursday'

when 6 then 'Friday'

when 7 then 'Saturday'

else 'Invalid day'

end

print @result

Repeative control structure:

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

\* User allows to create the loop

1.while loop

syntax:

while condition

begin

statement1

statement2

end

1 2 3 ....10

10 9 8 ....1

2 4 6 8 ....100

5 10 15 20 25 .....100

100 95 90 85 80 .....5

--2 4 6 8 ...100

declare @i int

set @i=2

while @i<=100

begin

print @i

set @i=@i+2

end

sql select statement:

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

select fieldname1,fieldname2,fieldname3 from tablename

where conditon;

tsql select statement syntax:

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

select @var1=fieldname1,@var2=fieldname2,

@var3=fieldname3 from tablename where <condition>

--WRITE A TSQL PROGRAM TO SELECT DEPT DETAILS FROM DEPT TABLE

DECLARE @A INT,@B VARCHAR(20),@C VARCHAR(20)

SELECT @A=DEPTNO,@B=DNAME,@C=LOC FROM DEPT WHERE DEPTNO=20

PRINT CAST(@A AS VARCHAR)+' '+@B+' '+@C

--tsql program to insert dept details into dept table and check the availabilty

set nocount on --stop the server msg

declare @a int,@b varchar(20),@c varchar(20)

set @a=50

set @b='HR'

set @c='HYD'

if not exists(Select deptno from dept where deptno=@a)

begin

insert into dept values(@a,@b,@c)

print '1 row inserted'

end

else

print 'Dept details exists'

--TSQL PROGRAM TO DELETE DEPT DETAILS FROM DEPT TABLE AND CHECK THE AVAILABILTY

set nocount on --stop the server msg

declare @a int

set @a=40

if exists(select deptno from dept where deptno=@a)

begin

delete from dept where deptno=@a

print cast(@@rowcount as varchar)+' = rows are deleted'

end

else

print 'Dept details does not exists'

--write a tsql program to modify the empsalary if it is working and valid increment >=500

set nocount on --stop the server msg

declare @empno int,@incr int

set @empno=7777

set @incr=1000

if not exists(select empno from emp where empno=@empno)

print 'Employee does not exists'

else if @incr <500

print 'Invalid Increment'

else

begin

update emp set sal=sal+@incr where empno=@empno

print cast(@@rowcount as varchar)+'= rows are updated'

end

CURSOR:

-------

1.A CURSOR IS A POINTER TO RESULTSET WHICH CONTAINS SET OF RECORDS RETURN BY THE SELECT STATEMENT.

2.USING THE CURSOR WE CAN PROCESS THE MULTIPLE RECORDS IN RESULTSET.

3.CURSOR WILL REDUCE NO.OF TRIPS TO DATABASE SERVER.

4.cursor memory alloted using "tempdb" database.

TO USE CURSOR FOLLOW BELOW STEPS:

1.DECLARE CURSOR

2.OPEN CURSOR

3.FETCH RECORDS FROM CURSOR

4.CLOSE CURSOR

5.DEALLOCATE CURSOR.

1.DECLARE THE CURSOR:

SYNTAX:

DECLARE <CURSORNAME> CURSOR[OPTIONS] FOR SELECT STATEMENT

EX:

DECLARE C1 CURSOR FOR SELECT \* FROM EMP

2.OPEN CURSOR

OPEN <CURSORNAME>

OPEN C1

3.FETCH RECORDS FROM CURSOR

FETCH NEXT|PRIOR|FIRST|LAST|ABSOLUTE N | RELATIVE N FROM

<CURSORNAME> INTO <VARIABLE>

EX:

FETCH NEXT FROM C1 INTO @ENAME,@ENO,@SAL

4.CLOSE CURSOR

SYNTAX:

CLOSE <CURSORNAME>

CLOSE C1

5.DEALLOCATE CURSOR

SYNTAX:

DEALLOCATE <CURSORNAME>

DEALLOCATE C1

@@FETCH\_STATUS either 0(true) or -1(false)

NOTE:

1. Any number of cursors can be created in a single program but with unique names.

2. Global variable @@FETCH\_STATUS plays an important role, where it returns boolean value i.e. returns 0

if it makes a fetch operation successfull over a cursor

else returns -1.

--write a tsql program to select all dept details from dept table

declare @a int,@b varchar(20),@c varchar(20)

declare c1 cursor for select \* from dept

open c1

fetch next from c1 into @a,@b,@c --start the loop

while (@@FETCH\_STATUS=0)

begin

print cast(@a as varchar)+' '+@b+' '+@c

fetch next from c1 into @a,@b,@c -- continue the loop

end

close c1

deallocate c1

--write a tsql program to select all ename,empno and salary from emp table

declare @ename varchar(20),@empno int,@sal int

declare @total int=0

declare c1 cursor for select ename,empno,sal from emp

open c1

fetch next from c1 into @ename,@empno,@sal --start the loop

while (@@FETCH\_STATUS=0)

begin

print @ename +' '+cast(@empno as varchar)+' '+cast(@sal as varchar)

set @total=@total+@sal

fetch next from c1 into @ename,@empno,@sal -- continue the loop

end

print 'Total salary='+cast(@total as varchar)

close c1

deallocate c1

--tsql program to select all ename and deptname from corresponding table

declare @ename varchar(20),@deptname varchar(20)

declare c1 cursor for select emp.ename,dept.dname from emp inner join dept on emp.deptno=dept.deptno

open c1

fetch next from c1 into @ename,@deptname --start the loop

while (@@FETCH\_STATUS=0)

begin

print @ename +' '+@deptname

fetch next from c1 into @ename,@deptname --continue the loop

end

close c1

deallocate c1

STUDENT

SNAME SNO S1 S2 S3

AAA 1001 89 78 100

BBB 1002 67 89 20

CCC 1003 78 87 88

DDD 1004 100 20 40

EEE 1005 78 67 99

Anil 1006 89 90 78

RESULT

SNO STOT SAVG SRES

drop table student

drop table result

create table student(sname varchar(20),sno int,s1 int,s2 int,s3 int)

insert into student values('aaa',1001,89,78,100)

insert into student values('bbb',1002,67,89,20)

insert into student values('ccc',1003,78,87,88)

insert into student values('ddd',1004,100,20,40)

insert into student values('uma',1005,90,78,90)

create table result(sno int,stot int,savg int,sres char(10))

--TSQL PROGRAM TO CREATE STUDENT MARK RESULT TABLE

truncate table result

declare @sno int,@s1 int,@s2 int,@s3 int,@stot int,@savg int,@sres varchar(10)

declare c1 cursor for select sno,s1,s2,s3 from student

open c1

fetch next from c1 into @sno,@s1,@s2,@s3 --start the loop

while (@@FETCH\_STATUS=0)

begin

set @stot=@s1+@s2+@s3

set @savg=@stot/3

if @s1>=40 and @s2>=40 and @s3>=40

set @sres='pass'

else

set @sres='fail'

insert into result values(@sno,@stot,@savg,@sres)

fetch next from c1 into @sno,@s1,@s2,@s3 -- continue the loop

end

close c1

deallocate c1

student

SNAME SNO S1 S2 S3

AAA 1001 89 78 100

BBB 1002 67 89 20

CCC 1003 78 87 88

DDD 1004 100 20 40

EEE 1005 78 67 99

result

SNO STOT SAVG SRES

1001 267 89 PASS

1002 176 58 FAIL

1003 253 84 PASS

1004 160 53 FAIL

1005 244 81 PASS

Assignment1:

-----------

employee

ename empno basicpay allow ded

aaa 1001 25000 1500 3500

bbb 1002 35000 2500 1200

ccc 1003 45000 7800 2500

ddd 1004 35000 10000 2500

eee 1005 12000 3500 4589

salary

empno grosspay netpay

grosspay=basicpay + allow

netpay=grosspay - ded

bank\_master

accno balance

1001 10000

1002 20000

1003 30000

1004 40000

1005 50000

bank\_Transaction

cname accno ttype amount

swathi 1003 db 2500

guna 1005 cr 5555

sekar 1001 db 2500

Note:

if ttype='cr'

update bank\_master set balance=balance + amount where accno=@accno

else

update bank\_master set balance=balance - amount where accno=@accno

--BANK

create table bank.banks(bank\_id int primary key,bank\_details varchar(23))

create table bank.addresses(Address\_id int primary key,line\_1 varchar(9),line\_2 varchar(9),town\_city varchar(20),state\_province\_country varchar(20),country varchar(30),other\_details varchar(100))

create table bank.ref\_branch\_types(branch\_type\_code int primary key,branch\_type\_description varchar(12),large\_urban varchar(30),small\_urban varchar(30))

create table bank.branches (branch\_id int primary key,address\_id int foreign key references bank.addresses(address\_id),bank\_id int foreign key references bank.Banks(bank\_id),branch\_type\_code int foreign key references bank.ref\_branch\_types(branch\_type\_code),branch\_details varchar(12) )

create table bank.customers(customer\_id int primary key,address\_id int foreign key references bank.addresses(address\_id),branch\_id int foreign key references bank.branches(branch\_id),personal\_details varchar(40),contact\_details varchar(20))

--ACCOUNT

create table account.ref\_account\_status(Account\_status\_code varchar(10) primary key,Account\_status\_description varchar(10))

create table account.ref\_account\_types(account\_type\_code varchar(10) primary key,account\_type\_description varchar(10))

create table account.Accounts(account\_number bigint primary key,Account\_status\_code varchar(10) foreign key references account.ref\_account\_status(Account\_status\_code),Account\_type\_code varchar(10) foreign key references account.ref\_account\_types(Account\_type\_code))

--TRANSACTIONS

create table ref\_transaction\_types

(

transaction\_type\_code int primary key,

transaction\_type\_description varchar(10)

)

create table transactions.transactions

(

Transaction\_id int primary key,

account\_number bigint foreign key references account.accounts(account\_number),

merchant\_id int,

transaction\_type\_code int foreign key references transactions.ref\_transaction\_types(transaction\_type\_code),

transaction\_date\_time datetime,

transaction\_amount int)njnjhn

)

CURSOR:

-------

1.A CURSOR IS A POINTER TO RESULTSET WHICH CONTAINS SET OF RECORDS RETURN BY THE SELECT STATEMENT.

2.USING THE CURSOR WE CAN PROCESS THE MULTIPLE RECORDS IN RESULTSET.

3.CURSOR WILL REDUCE NO.OF TRIPS TO DATABASE SERVER.

4.cursor memory alloted using "tempdb" database.

TO USE CURSOR FOLLOW BELOW STEPS:

1.DECLARE CURSOR

2.OPEN CURSOR

3.FETCH RECORDS FROM CURSOR

4.CLOSE CURSOR

5.DEALLOCATE CURSOR.

1.DECLARE THE CURSOR:

SYNTAX:

DECLARE <CURSORNAME> CURSOR[OPTIONS] FOR SELECT STATEMENT

EX:

DECLARE C1 CURSOR FOR SELECT \* FROM EMP

2.OPEN CURSOR

OPEN <CURSORNAME>

OPEN C1

3.FETCH RECORDS FROM CURSOR

FETCH NEXT|PRIOR|FIRST|LAST|ABSOLUTE N | RELATIVE N FROM

<CURSORNAME> INTO <VARIABLE>

EX:

FETCH NEXT FROM C1 INTO @ENAME,@ENO,@SAL

4.CLOSE CURSOR

SYNTAX:

CLOSE <CURSORNAME>

CLOSE C1

5.DEALLOCATE CURSOR

SYNTAX:

DEALLOCATE <CURSORNAME>

DEALLOCATE C1

@@FETCH\_STATUS either 0(true) or -1(false)

NOTE:

1. Any number of cursors can be created in a single program but with unique names.

2. Global variable @@FETCH\_STATUS plays an important role, where it returns boolean value i.e. returns 0

if it makes a fetch operation successfull over a cursor

else returns -1.

--write a tsql program to select all dept details from dept table

declare @a int,@b varchar(20),@c varchar(20)

declare c1 cursor for select \* from dept

open c1

fetch next from c1 into @a,@b,@c --start the loop

while (@@FETCH\_STATUS=0)

begin

print cast(@a as varchar)+' '+@b+' '+@c

fetch next from c1 into @a,@b,@c -- continue the loop

end

close c1

deallocate c1

--write a tsql program to select all ename,empno and salary from emp table

declare @ename varchar(20),@empno int,@sal int

declare @total int=0

declare c1 cursor for select ename,empno,sal from emp

open c1

fetch next from c1 into @ename,@empno,@sal --start the loop

while (@@FETCH\_STATUS=0)

begin

print @ename +' '+cast(@empno as varchar)+' '+cast(@sal as varchar)

set @total=@total+@sal

fetch next from c1 into @ename,@empno,@sal -- continue the loop

end

print 'Total salary='+cast(@total as varchar)

close c1

deallocate c1

--tsql program to select all ename and deptname from corresponding table

declare @ename varchar(20),@deptname varchar(20)

declare c1 cursor for select emp.ename,dept.dname from emp inner join dept on emp.deptno=dept.deptno

open c1

fetch next from c1 into @ename,@deptname --start the loop

while (@@FETCH\_STATUS=0)

begin

print @ename +' '+@deptname

fetch next from c1 into @ename,@deptname --continue the loop

end

close c1

deallocate c1

STUDENT

SNAME SNO S1 S2 S3

AAA 1001 89 78 100

BBB 1002 67 89 20

CCC 1003 78 87 88

DDD 1004 100 20 40

EEE 1005 78 67 99

Anil 1006 89 90 78

RESULT

SNO STOT SAVG SRES

drop table student

drop table result

create table student(sname varchar(20),sno int,s1 int,s2 int,s3 int)

insert into student values('aaa',1001,89,78,100)

insert into student values('bbb',1002,67,89,20)

insert into student values('ccc',1003,78,87,88)

insert into student values('ddd',1004,100,20,40)

insert into student values('uma',1005,90,78,90)

create table result(sno int,stot int,savg int,sres char(10))

--TSQL PROGRAM TO CREATE STUDENT MARK RESULT TABLE

truncate table result

declare @sno int,@s1 int,@s2 int,@s3 int,@stot int,@savg int,@sres varchar(10)

declare c1 cursor for select sno,s1,s2,s3 from student

open c1

fetch next from c1 into @sno,@s1,@s2,@s3 --start the loop

while (@@FETCH\_STATUS=0)

begin

set @stot=@s1+@s2+@s3

set @savg=@stot/3

if @s1>=40 and @s2>=40 and @s3>=40

set @sres='pass'

else

set @sres='fail'

insert into result values(@sno,@stot,@savg,@sres)

fetch next from c1 into @sno,@s1,@s2,@s3 -- continue the loop

end

close c1

deallocate c1

student

SNAME SNO S1 S2 S3

AAA 1001 89 78 100

BBB 1002 67 89 20

CCC 1003 78 87 88

DDD 1004 100 20 40

EEE 1005 78 67 99

result

SNO STOT SAVG SRES

1001 267 89 PASS

1002 176 58 FAIL

1003 253 84 PASS

1004 160 53 FAIL

1005 244 81 PASS

Assignment1:

-----------

employee

ename empno basicpay allow ded

aaa 1001 25000 1500 3500

bbb 1002 35000 2500 1200

ccc 1003 45000 7800 2500

ddd 1004 35000 10000 2500

eee 1005 12000 3500 4589

salary

empno grosspay netpay

grosspay=basicpay + allow

netpay=grosspay - ded

bank\_master

accno balance

1001 10000

1002 20000

1003 30000

1004 40000

1005 50000

bank\_Transaction

cname accno ttype amount

swathi 1003 db 2500

guna 1005 cr 5555

sekar 1001 db 2500

mano 1004 db 3500

Note:

if ttype='cr'

update bank\_master set balance=balance + amount where accno=@accno

else

update bank\_master set balance=balance - amount where accno=@accno

--Write tsql program to select all ename and deptname from corresponding table

declare @ename varchar(20),@dname varchar(20)

declare c1 cursor for select emp.ename,dept.dname from emp inner join dept on emp.deptno=dept.deptno

open c1

fetch next from c1 into @ename,@dname --start the loop

while(@@FETCH\_STATUS=0)

begin

print @ename +' '+@dname

fetch next from c1 into @ename,@dname --continue the loop

end

close c1

deallocate c1

CLARK ACCOUNTING

KING ACCOUNTING

MILLER ACCOUNTING

FORD RESEARCH

ADAMS RESEARCH

SCOTT RESEARCH

SMITH RESEARCH

JONES RESEARCH

MARTIN SALES

BLAKE SALES

ALLEN SALES

WARD SALES

TURNER SALES

JAMES SALES

--Write tsql program to select all ename and deptname from corresponding table in reverse order

declare @ename varchar(20),@dname varchar(20)

declare c1 cursor scroll for select emp.ename,dept.dname from emp inner join dept on emp.deptno=dept.deptno

open c1

fetch last from c1 into @ename,@dname --start the loop

while(@@FETCH\_STATUS=0)

begin

print @ename +' '+@dname

fetch prior from c1 into @ename,@dname --continue the loop

end

close c1

deallocate c1

JAMES SALES

TURNER SALES

WARD SALES

ALLEN SALES

BLAKE SALES

MARTIN SALES

JONES RESEARCH

SMITH RESEARCH

SCOTT RESEARCH

ADAMS RESEARCH

FORD RESEARCH

MILLER ACCOUNTING

KING ACCOUNTING

CLARK ACCOUNTING

SCROLL OPTION

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

NEXT - the default cursor fetch option.FETCH NEXT returns the next row after the current row.

PRIOR - returns the previous row before the current row.

FIRST - returns the first row in the cursor.

LAST - returns the last row in the cursor.

ABSOLUTE N - returns the nth row in the cursor first position. (2 records)

RELATIVE N - returns the nth row in the cursor current position. (2 ,4,6,8....)

--display all emp details in reverse order

--display all ename,empno and salary from emp table in reverse order

set nocount on --stop the server msg

declare @ename varchar(20),@empno int,@sal int

declare c1 cursor scroll for select ename,empno,sal from emp

open c1

fetch last from c1 into @ename,@empno,@sal --start the loop

while(@@FETCH\_STATUS=0)

begin

print @ename +' '+cast(@empno as varchar)+' '+cast(@sal as varchar)

fetch prior from c1 into @ename,@empno,@sal --continue the loop

end

close c1

deallocate c1

--display all emp details in Even series records

set nocount on --stop the server msg

declare @ename varchar(20),@empno int,@sal int

declare c1 cursor scroll for select ename,empno,sal from emp

open c1

fetch absolute 2 from c1 into @ename,@empno,@sal --start the loop

while(@@FETCH\_STATUS=0)

begin

print @ename +' '+cast(@empno as varchar)+' '+cast(@sal as varchar)

fetch relative 2 from c1 into @ename,@empno,@sal --continue the loop

end

close c1

deallocate c1

--display all emp details in odd series records

declare @ename varchar(20),@empno int,@sal int

declare c1 cursor scroll for select ename,empno,sal from emp

open c1

fetch absolute 1 from c1 into @ename,@empno,@sal --start the loop

while(@@FETCH\_STATUS=0)

begin

print @ename+' '+cast(@empno as varchar)+' '+cast(@sal as varchar)

fetch relative 2 from c1 into @ename,@empno,@sal -- continue the loop

end

close c1

deallocate c1

--display all emp details in 4th series records

declare @ename varchar(20),@empno int,@sal int

declare c1 cursor scroll for select ename,empno,sal from emp

open c1

fetch absolute 4 from c1 into @ename,@empno,@sal --start the loop

while(@@FETCH\_STATUS=0)

begin

print @ename+' '+cast(@empno as varchar)+' '+cast(@sal as varchar)

fetch relative 4 from c1 into @ename,@empno,@sal -- continue the loop

end

close c1

deallocate c1

Types of Cursor:

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

1.Static cursor

2.Dynamic cursor (by default)

Static cursor:

\* If the cursor is declared with static, any changes make to the base table changes are not reflected to result set.

Dynamic cursor :

\* If the cursor is declared with dynamic so any changes make to the base table automatically reflected to result set.

declare @sal int

declare c1 cursor static/ dynamic for select sal from emp

where empno=7788

open c1

update EMP set SAL=SAL+1000 where EMPNO=7788

fetch next from c1 into @sal

print @sal

close c1

deallocate c1

Advantage of Cursor:

1.Row-by-row operations:

2.Efficiency: The query optimizer automatically selects theappropriate query plan, so the developer does not need to design a complex algorithm to access the required data.

3.Adaptability: As data changes or indexes are added or

dropped, the query optimizer automatically adapts its

behavior by using alternative plans.

STRUCTURED EXCEPTION HANDLING:

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

Error Handling

1.Compile time error

2.Runtime error -- Exception

In SQL Server 2012 exception handling is carried out using Strucuted Exception, which supports to write 2 blocks.

1. TRY : It is used to monitor all those instructions in which run time errors are expected.

2. CATCH : It is used to catch the thrown exceptions. This block is executed only when a exception is raised.

Syntax:

BEGIN TRY

statements

END TRY

BEGIN CATCH

statements

END CATCH

begin try

declare @a int,@b int,@c int

set @a=100

set @b=0

set @c=@a/@b

print 'Quotient values='+cast(@c as varchar)

end try

begin catch

print error\_message()

print error\_number()

print error\_line()

print error\_state()

end catch

Note:

Following are the functions is used to display information about errors.

ERROR\_NUMBER() -returns the number of the error.

ERROR\_STATE() -returns the error state number.

ERROR\_PROCEDURE()- returns the name of the stored procedure or trigger where the error occurred.

ERROR\_LINE() - returns the line number inside the routine that caused the error.

ERROR\_MESSAGE() - returns the complete text of the error message.

DIFFERENCE BETWEEN ANONYMOUS BLOCKS AND NAMED BLOCKS or stored block

ANONYMOUS NAMED/STORED

1.BLOCKS WITHOUT NAME 1.BLOCKS WITH NAME

2.ANNONYMOUS BLOCKS ARE 2. BUT THE NAMED BLOCKS ARE SAVED IN

SAVED IN "OPERATING SYSTEM FILE" "DATABASESERVER".

3.THESE BLOCKS ARE NOT SECURED. 3.BUT NAMED BLOCKS ARE SECURED.

4.THESE BLOCKS CAN'T BE REUSED. 4.BUT NAMED BLOCKS CAN BE REUSED.

5.EVERY TIME COMPILE & EXECUTED.5.NAMED BLOCKS ARE ONE TIME

COMPILATION AND EVERY TIME EXECUTION.

Sub Programs:

It is a process of splitting a large application program into small modules or blocks.

SQL Server supports to write a sub program based on the following concepts;

1. Stored Procedures

2. Stored Functions

3. Stored Triggers

Advantages of Sub programs:

1. Provides security.

2. It improves performance

3. Reduces Network Traffic

4. Readability gets increases

5. Code Reusability

6. Error Detection and modification is quite easy

7. Extensibility - It will allow a user to

increase or decrease the code.

Stored Procedures:

Procedure is a self contained program or predefined program, which may or may not return value.

SQL server supports to work with the

following types of procedures;

i. Predefined Procedures (only db level functions)

sp\_help - return table details

sp\_helpdb - return database details

sp\_spaceused -- current database size

sp\_renamedb- rename datbase

sp\_rename - rename table

sp\_helptext - return procedure details

ii. User Defined Procedures

iii.Extended\_procedures (xp\_)

xp\_fixedrives (mail server and os level)

User Defined Procedures:

It is a stored block database object, which resides in database server.

It is a self contained program or predefined program which may or may not return value.

It can be created with and without arguments.

Data can be sent to procedures to process it using Input Arguments and few values can be taken back to the main program using OUTPUT arguments.

A procedure can be set with 1024 Input and 1024 Output Arguments.

When a procedure is set with OUTPUT arguments then it is essential that at sending argument OUTPUT should be specified.

Procedure set with input arguments will work for call by value and output arguments will work for call by reference.

Syntax:

CREATE PROC / PROCEDURE proc\_name([list\_of\_args])

[WITH ENCRYPTION]

AS

BEGIN

[declaration block]

execution block

END

A Procedure can be executed in 2 ways;

1. using EXEC command

Syntax:

[EXEC] proc\_name [list\_of\_args]

[EXECUTE] proc\_name [list\_of\_args]

EXAMPLES:

-- Create a procedure with no arguments and retrieve the data of emp table?

create procedure empret

as

begin

select \* from emp

end

--how to invoke stored procedure

exec empret

2.--create a procedure to accept deptno as argument and return corresponding emp details

create procedure empret2(@a int)

as

begin

select \* from emp where deptno =@a

end

exec empret2 10

exec empret2 20

exec empret2 30

exec empret2

--create a procedure to accept deptno as argument and return corresponding emp details

--II method

--create a procedure to accept deptno as argument and return corresponding emp details

alter procedure empret2(@a int=null)

as

begin

if @a is null

select \* from emp

else

select \* from emp where deptno =@a

end

exec empret2 10

exec empret2 20

exec empret2 30

exec empret2

--how to invoke stored procedure

exec emprt2 10

exec emprt2 20

exec emprt2 30

exec emprt2

--create a procedure to accept 2 deptno as argument and return correponding emp details

create procedure empret3(@a int=null,@b int=null)

as

begin

if @a is null and @b is null

select \* from emp

else if @a is null

select \* from emp where deptno=@b

else if @b is null

select \* from emp where deptno=@a

else

select \* from emp where deptno in(@a,@b)

end

exec empret3 10,20

exec empret3 10

exec empret3 20

exec empret3

exec emprt3 10

exec emprt3 10,20

exec emprt3

--create a procedure to insert the records into dept table and check validation

create procedure insrec1(@a int,@b varchar(20),@c varchar(20))

as

begin

set nocount on --stop the server msg

if not exists(select deptno from dept where deptno=@a)

begin

insert into dept values(@a,@b,@c)

print cast(@@rowcount as varchar)+' rows are inserted'

end

else

print 'dept details exists'

end

--how to invoke stored procedure

exec insrec1 50,'hr','hyd'

--create a procedure to delete dept details from dept table and check the validation.

create procedure delrec1(@a int)

as

begin

set nocount on --stop the server msg

if exists(select deptno from dept where deptno=@a)

begin

delete from dept where deptno=@a

print cast(@@rowcount as varchar)+'=rows are deleted'

end

else

print 'dept details does not exists'

end

exec delrec1 50

exec delrec1 40

--Create a procedure to Update the salary of employee if it is valid increment of more than 500?

create procedure updrow(@empno int,@incr int)

as

begin

set nocount on --stop the server msg

if exists(select empno from emp where empno=@empno)

begin

if @incr >=500

begin

update emp set sal=sal+@incr where empno=@empno

print cast(@@rowcount as varchar)+'=rows are updated'

end

else

print 'Invalid increment'

end

else

print 'Employee does not exists'

end

exec updrow 4444,1000 --Employee does not exists

exec updrow 7788,250 --Invalid increment

exec updrow 7788,1000 --updated sucessfully

7. Create a procedure to extract only 10 digit phone number?

create PROCEDURE P1(@A BIGINT)

AS

BEGIN

declare @result varchar(20)

set @result=RIGHT(cast(@a as varchar),10)

print @result

end

exec p1 1010034949489494

8. Create a procedure to display holiday list?

stdate: 04-01-2018 enddate 04-30-2018

create procedure holiday(@stdate datetime,@enddate datetime)

as

begin

while @stdate <=@enddate

begin

if datename(dw,@stdate) in('Saturday','Sunday') --day of week

print convert(varchar,@stdate,105)

set @stdate=@stdate +1

end

end

exec holiday '03-01-2015','12-31-2015'

--working day list

alter procedure workingday(@stdate datetime,@enddate datetime)

as

begin

while @stdate <=@enddate

begin

if datename(dw,@stdate) not in('Saturday','Sunday') --day of week

print convert(varchar,@stdate,105)

set @stdate=@stdate +1

end

end

exec workingday '03-01-2015','12-31-2015'

--create a procedure to accept two input arguments and return 2 output arguments.

create procedure arithmetic(@a int,@b int,

@c int output,@d int output)

as

begin

set @c=@a+@b

set @d=@a\*@b

end

-how to invoke stored procedure

declare @add int,@mul int

exec arithmetic 100,20,@add output,@mul output

print @add

print @mul

--example 2

create procedure p11(@a int,@s int output,@c int output)

as

begin

set @s=@a\*@a

set @c=@a\*@a\*@a

end

--execution

declare @s int,@c int

exec p11 10,@s output,@c output

print @s

print @c

1. Create a procedure that takes empno as input and displays ename,job,sal,deptno of that employee?

--Different types of Print Statement

CREATE PROC ERET1(@ENO INT)

AS

BEGIN

SELECT ENAME,JOB,SAL,DEPTNO FROM EMP WHERE EMPNO=@ENO

END

EXEC ERET1 7788

EXEC ERET1 7369

OR

CREATE PROC ERET2(@ENO INT)

AS

BEGIN

DECLARE @EN VARCHAR(12),@J VARCHAR(14),@PAY INT,@DNO INT

SELECT @EN=ENAME,@J=JOB,@PAY=SAL,@DNO=DEPTNO FROM EMP WHERE EMPNO=@ENO

PRINT @ENO

PRINT @EN

PRINT @J

PRINT @PAY

PRINT @DNO

END

EXEC ERET2 7788

OR

CREATE PROC ERET3(@ENO INT,@EN VARCHAR(12) OUTPUT,@J VARCHAR(12) OUTPUT,@PAY INT OUTPUT,@DNO INT OUTPUT)

AS

BEGIN

SELECT @EN=ENAME,@J=JOB,@PAY=SAL,@DNO=DEPTNO FROM EMP WHERE EMPNO=@ENO

END

--how to execute stored prcoedure?

DECLARE @EC INT,@N VARCHAR(12),@D VARCHAR(12),@P INT,@DC INT

SET @EC=7902

EXEC ERET3 @EC,@N OUTPUT,@D OUTPUT,@P OUTPUT,@DC OUTPUT

PRINT @EC

PRINT @N

PRINT @D

PRINT @P

PRINT @DC

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

line - while loop

print char - while loop

declare @i int

set @i=1

while @i<=5

begin

print cast(replicate(@i,@i) as varchar)

set @i=@i+1

end

display this format

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

DYNAMIC QUERIES:

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

A query in a block is provided with input

at the execution of a program,

so that dynamically changes can be made to the query,and will generate the different output.

-- Retrieving the data from any table:

create procedure rettable(@tn varchar(20))

as

begin

exec('select \* from '+@tn)

end

exec rettable 'emp'

exec rettable 'dept'

exec rettable 'salgrade'

exec rettable 'emp1'

--2. Create a table using procedure with passing table names at runtime:

CREATE PROC CT(@TN VARCHAR(12))

AS

BEGIN

EXEC ('CREATE TABLE '+@TN+'(ENO INT,EN VARCHAR(12))')

END

exec ct 'm1'

exec ct 'm2'

exec ct 'm3'

select \* from m1

exec ct 'm4'

3. Creating a table with table names, column names, data types dynamically?

CREATE PROC CT1(@TN VARCHAR(12),@COL VARCHAR(12),@DT VARCHAR(12))

AS

BEGIN

EXEC ('CREATE TABLE '+@TN+'('+@COL+' '+@DT+')')

END

EXEC CT1 'T1','ENO','INT'

EXEC CT1 'T2','RNO','SMALLINT'

4. Adding a new column in a table?

CREATE PROC AT1(@TN VARCHAR(12))

AS

BEGIN

EXEC ('ALTER TABLE '+@TN+' ADD NAME VARCHAR(12)')

END

EXEC AT1 T1

5. Altering the size of existing column:

CREATE PROC AT2(@TN VARCHAR(12))

AS

BEGIN

EXEC ('ALTER TABLE '+@TN+' ALTER COLUMN NAME VARCHAR(20)')

END

EXEC AT2 T1

6. Droping a table:

CREATE PROC DT(@TN VARCHAR(12))

AS

BEGIN

EXEC ('DROP TABLE '+@TN)

END

EXEC DT E1

EXEC DT E2

--how to know howmany tables are created?

select name from sys.tables

--create a procedure to accept table id through argument and return corresponding table data

step1 : create table tablelist(tid int identity,tname varchar(20))

step2 : insert into tablelist select name from sys.tables

select \* from tablelist

create procedure rettab1(@tid int)

as

begin

declare @tname varchar(20)

select @tname=tname from tablelist where tid=@tid

exec ('select \* from '+@tname)

end

exec rettab 1

exec rettab 2

exec rettab 3

exec rettab 30

Here are some instruction when creating a store procedure to increase speed

or

Here are the some good tips when creating a store procedure

\* Use SQL keyword in capital letters to increase readability.

\* Use few possible variables to increase cache memory.

\* Try to avoid dynamic queries if we are not using dynamic query there is no recompilation of execution plan but on the other side if we are using dynamic query every time we need recompile of plan.

\* Use SET NOCOUNT ON this will helps us to get number of row effected without network traffic.

\* In Select and Set use select to assign value to variable

it is much faster than multiple set statement.

set @a=100

set @b=200

set @c=@a+@b

set @d=@a-@b

or

select @a=100,@b=200,@c=@a+@b,@d=@a-@b

\* In CAST and CONVERT always try to use CAST

\* Avoid Distinct and Order by clause.These class needs extra space.

\* Try to avoid cursor.

\* Avoid correlated sub queries.

\* Avoid select \* into for large tables it locks the system objects.

select ename,empno,sal from emp

\* Avoid temporary tables because it will recompile the procedure.

\* Use schema name with object name: The object name is qualified if used with schema name. Schema name should be used with the stored procedure name and with all objects referenced inside the stored procedure. This help in directly finding the complied plan instead of searching the objects in other possible schema.

select \* from dbo.emp

select \* from dbo.dept

select \* from dbo.salgrade

Modify the procedure:

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

alter procedure <procedure\_name>(list of args)

as

begin

declaration statement

executable statement

end

To display all predefined, userdefined stored procedures:

SP\_STORED\_PROCEDURES

To display all the created user defined procedure names:

SELECT NAME FROM SYSOBJECTS WHERE XTYPE='P'

C = CHECK constraint

D = Default or DEFAULT constraint

F = FOREIGN KEY constraint

L = Log

P = Stored procedure

PK = PRIMARY KEY constraint

RF = Replication filter stored procedure

S = System table

TR = Trigger

U = User table

UQ = UNIQUE constraint

V = View

X = Extended stored procedure

FN = FUNCTION

To display the code of a user defined stored procedure:

SP\_HELPTEXT 'proc\_name'

To display the information about procedure:

SP\_HELP 'proc\_name'

--how to know howmany depends objects are created?

sp\_depends 'tablename' (Real time usage)

drop procedure procedurename

drop procedure empret

Extended Stored Procedures:

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

A stored procedure contains T-SQL statements and C/C++ code compiled into .dll (Dynamic link library) file is called extended stored procedure.

Every extended stored procedure starts with XP\_

All extended stored procedures are stored in MASTER database.

Extended procedures are used to perform operations in OS and Mail Server.

XP\_CMDSHELL 'os command' ---- It is used to communicate

with operating system to execute commands.

To work with this extended procedure, it is essential that it should be enabled with the following procedure;

Example:

EXEC XP\_CMDSHELL 'MD C:\SQLS'

EXEC XP\_CMDSHELL 'DIR C:\'

EXEC XP\_SENDMAIL |

EXEC XP\_READMAIL | to interact with mail server.

xp\_cmdshell 'dir c:\'

xp\_cmdshell 'md c:\shruthi'

how to check the drive free space?

xp\_fixeddrives

xp\_fixeddrives

xp\_cmdshell 'md c:\jayasri'

xp\_cmdshell 'dir c:\'

xp\_cmdshell 'rd c:\jayasri'

STORED FUNCTIONS

(USER DEFINED FUNCTIONS)

Function is defined as "self contained or predefined program which returns max of 1 value".

Returned value can of int, varchar, datetime.

User defined functions are classified into 2 types;

I. Scalar valued functions (which returns 1 value)

II. Table valued functions (which returns multiple rows)

difference between procedure and function:

1)Procedure can return zero or n values whereas function can return one value which is mandatory.

2)Procedures can have input,output parameters for it whereas functions can have only input parameters.

3)Procedure allow select as well as DML (Insert,update,delete) statement in it whereas function allow only select statement in it.

4)Functions can be called from procedure whereas procedures cannot be called from function.

5)Exception can be handled by try-catch block in a procedure whereas try-catch block cannot be used in a function.

6)We can go for transaction management in procedure whereas we can't go in function.

7)Procedures can not be utilized in a select statement whereas function can be embedded in a select statement.

ex:

select sum(sal) from emp

select avg(sal) from emp

select len(ename) from emp

exec procedurename

I. Scalar valued functions : These functions are stored as database objects in a database server and can return max of 1 value. It takes the input thru arguments and returns 1 value after processing those arguments.

syntax:

CREATE FUNCTION func\_name(list\_of\_args)

RETURNS data\_type[(size)]

[WITH ENCRYPTION]

AS

BEGIN

[declaration part]

[Execution part]

RETURN var/val

END

--how to invoke userdefined function?

select username.functionname()

select dbo.fun1()

select dbo.fun2()

select fun1() --invalid

examples:

1.create a function that takes 2 values as input and returns sum of 2 values?

CREATE FUNCTION sumnos1(@a INT,@b INT)

RETURNS INT

AS

BEGIN

DECLARE @c INT

SET @c=@a+@b

RETURN @c

END

select dbo.sumnos1(100,200) --correct

select sumnos1(200,300) --incorrect

2.create a function to accept empno as argument and return corresponding ename

create function empretfn(@empno int)

returns varchar(20)

as

begin

declare @ename varchar(20)

select @ename=ename from emp where empno=@empno

return @ename

end

--how to invoke stored function

select dbo.empretfn(7788)

select dbo.empretfn(7902)

3.create a function to calculate factorial value of n numbers?

1!= 1

2!= 1x 2=2

3!=1x2x3=6

4!=1x2x3x4=24

5!=1x2x3x4x5=120

n!=1x2x3x4x5......xn

create function fact(@n int)

returns int

as

begin

declare @i int,@f int

set @i=1

set @f=1

while @i<=@n

begin

set @f=@f\*@i

set @i=@i+1

end

return @f

end

select dbo.fact1(5)

--create a function to display yesterday date

create function yesterday()

returns datetime

as

begin

declare @d1 datetime

select @d1=getdate()-1

return @d1

end

select dbo.yesterday()

--create a function to display nextday

create function nextday()

returns varchar(20)

as

begin

declare @day varchar(20)

set @day=datename(dw,getdate()+1)

return @day

end

select dbo.nextday()

create function months\_between(@d1 datetime,@d2 datetime)

returns int

as

begin

declare @diff int

set @diff=datediff(mm,@d1,@d2)

return @diff

end

select dbo.months\_between(getdate(),'06-18-2016')

--Write a tsql program to get the years, months and days between two dates.

Declare @dateofbirth datetime

Declare @currentdatetime datetime

Declare @years varchar(40)

Declare @months varchar(30)

Declare @days varchar(30)

set @dateofbirth='1988-12-06'--birthdate

set @currentdatetime =getdate()--current datetime

select @years=datediff(year,@dateofbirth,@currentdatetime)-- To find Years

select @months=datediff(month,@dateofbirth,@currentdatetime)-(datediff(year,@dateofbirth,@currentdatetime)\*12)

-- To Find Months

select @days= datepart(d,@dateofbirth) - datepart(d,@currentdatetime)-- To Find Days

select @years +' years, ' +@months +' months, '+@days +' days' asYearMonthDay

DYNAMIC QUERIES:

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

A query in a block is provided with input

at the execution of a program,

so that dynamically changes can be made to the query,and will generate the different output.

-- Retrieving the data from any table:

create procedure rettable(@tn varchar(20))

as

begin

exec('select \* from '+@tn)

end

exec rettable 'emp'

exec rettable 'dept'

exec rettable 'salgrade'

exec rettable 'emp1'

--2. Create a table using procedure with passing table names at runtime:

CREATE PROC CT(@TN VARCHAR(12))

AS

BEGIN

EXEC ('CREATE TABLE '+@TN+'(ENO INT,EN VARCHAR(12))')

END

exec ct 'm1'

exec ct 'm2'

exec ct 'm3'

select \* from m1

exec ct 'm4'

3. Creating a table with table names, column names, data types dynamically?

CREATE PROC CT1(@TN VARCHAR(12),@COL VARCHAR(12),@DT VARCHAR(12))

AS

BEGIN

EXEC ('CREATE TABLE '+@TN+'('+@COL+' '+@DT+')')

END

EXEC CT1 'T1','ENO','INT'

EXEC CT1 'T2','RNO','SMALLINT'

4. Adding a new column in a table?

CREATE PROC AT1(@TN VARCHAR(12))

AS

BEGIN

EXEC ('ALTER TABLE '+@TN+' ADD NAME VARCHAR(12)')

END

EXEC AT1 T1

5. Altering the size of existing column:

CREATE PROC AT2(@TN VARCHAR(12))

AS

BEGIN

EXEC ('ALTER TABLE '+@TN+' ALTER COLUMN NAME VARCHAR(20)')

END

EXEC AT2 T1

6. Droping a table:

CREATE PROC DT(@TN VARCHAR(12))

AS

BEGIN

EXEC ('DROP TABLE '+@TN)

END

EXEC DT E1

EXEC DT E2

--how to know howmany tables are created?

select name from sys.tables

--create a procedure to accept table id through argument and return corresponding table data

step1 : create table tablelist(tid int identity,tname varchar(20))

step2 : insert into tablelist select name from sys.tables

select \* from tablelist

create procedure rettab1(@tid int)

as

begin

declare @tname varchar(20)

select @tname=tname from tablelist where tid=@tid

exec ('select \* from '+@tname)

end

exec rettab 1

exec rettab 2

exec rettab 3

exec rettab 30

Here are some instruction when creating a store procedure to increase speed

or

Here are the some good tips when creating a store procedure

\* Use SQL keyword in capital letters to increase readability.

\* Use few possible variables to increase cache memory.

\* Try to avoid dynamic queries if we are not using dynamic query there is no recompilation of execution plan but on the other side if we are using dynamic query every time we need recompile of plan.

\* Use SET NOCOUNT ON this will helps us to get number of row effected without network traffic.

\* In Select and Set use select to assign value to variable

it is much faster than multiple set statement.

set @a=100

set @b=200

set @c=@a+@b

set @d=@a-@b

or

select @a=100,@b=200,@c=@a+@b,@d=@a-@b

\* In CAST and CONVERT always try to use CAST

\* Avoid Distinct and Order by clause.These class needs extra space.

\* Try to avoid cursor.

\* Avoid correlated sub queries.

\* Avoid select \* into for large tables it locks the system objects.

select ename,empno,sal from emp

\* Avoid temporary tables because it will recompile the procedure.

\* Use schema name with object name: The object name is qualified if used with schema name. Schema name should be used with the stored procedure name and with all objects referenced inside the stored procedure. This help in directly finding the complied plan instead of searching the objects in other possible schema.

select \* from dbo.emp

select \* from dbo.dept

select \* from dbo.salgrade

Modify the procedure:

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

alter procedure <procedure\_name>(list of args)

as

begin

declaration statement

executable statement

end

To display all predefined, userdefined stored procedures:

SP\_STORED\_PROCEDURES

To display all the created user defined procedure names:

SELECT NAME FROM SYSOBJECTS WHERE XTYPE='P'

C = CHECK constraint

D = Default or DEFAULT constraint

F = FOREIGN KEY constraint

L = Log

P = Stored procedure

PK = PRIMARY KEY constraint

RF = Replication filter stored procedure

S = System table

TR = Trigger

U = User table

UQ = UNIQUE constraint

V = View

X = Extended stored procedure

FN = FUNCTION

To display the code of a user defined stored procedure:

SP\_HELPTEXT 'proc\_name'

To display the information about procedure:

SP\_HELP 'proc\_name'

--how to know howmany depends objects are created?

sp\_depends 'tablename' (Real time usage)

drop procedure procedurename

drop procedure empret

Extended Stored Procedures:

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

A stored procedure contains T-SQL statements and C/C++ code compiled into .dll (Dynamic link library) file is called extended stored procedure.

Every extended stored procedure starts with XP\_

All extended stored procedures are stored in MASTER database.

Extended procedures are used to perform operations in OS and Mail Server.

XP\_CMDSHELL 'os command' ---- It is used to communicate

with operating system to execute commands.

To work with this extended procedure, it is essential that it should be enabled with the following procedure;

Example:

EXEC XP\_CMDSHELL 'MD C:\SQLS'

EXEC XP\_CMDSHELL 'DIR C:\'

EXEC XP\_SENDMAIL |

EXEC XP\_READMAIL | to interact with mail server.

xp\_cmdshell 'dir c:\'

xp\_cmdshell 'md c:\shruthi'

how to check the drive free space?

xp\_fixeddrives

xp\_fixeddrives

xp\_cmdshell 'md c:\jayasri'

xp\_cmdshell 'dir c:\'

xp\_cmdshell 'rd c:\jayasri'

STORED FUNCTIONS

(USER DEFINED FUNCTIONS)

Function is defined as "self contained or predefined program which returns max of 1 value".

Returned value can of int, varchar, datetime.

User defined functions are classified into 2 types;

I. Scalar valued functions (which returns 1 value)

II. Table valued functions (which returns multiple rows)

difference between procedure and function:

1)Procedure can return zero or n values whereas function can return one value which is mandatory.

2)Procedures can have input,output parameters for it whereas functions can have only input parameters.

3)Procedure allow select as well as DML (Insert,update,delete) statement in it whereas function allow only select statement in it.

4)Functions can be called from procedure whereas procedures cannot be called from function.

5)Exception can be handled by try-catch block in a procedure whereas try-catch block cannot be used in a function.

6)We can go for transaction management in procedure whereas we can't go in function.

7)Procedures can not be utilized in a select statement whereas function can be embedded in a select statement.

ex:

select sum(sal) from emp

select avg(sal) from emp

select len(ename) from emp

exec procedurename

I. Scalar valued functions : These functions are stored as database objects in a database server and can return max of 1 value. It takes the input thru arguments and returns 1 value after processing those arguments.

syntax:

CREATE FUNCTION func\_name(list\_of\_args)

RETURNS data\_type[(size)]

[WITH ENCRYPTION]

AS

BEGIN

[declaration part]

[Execution part]

RETURN var/val

END

--how to invoke userdefined function?

select username.functionname()

select dbo.fun1()

select dbo.fun2()

select fun1() --invalid

examples:

1.create a function that takes 2 values as input and returns sum of 2 values?

CREATE FUNCTION sumnos1(@a INT,@b INT)

RETURNS INT

AS

BEGIN

DECLARE @c INT

SET @c=@a+@b

RETURN @c

END

select dbo.sumnos1(100,200) --correct

select sumnos1(200,300) --incorrect

2.create a function to accept empno as argument and return corresponding ename

create function empretfn(@empno int)

returns varchar(20)

as

begin

declare @ename varchar(20)

select @ename=ename from emp where empno=@empno

return @ename

end

--how to invoke stored function

select dbo.empretfn(7788)

select dbo.empretfn(7902)

3.create a function to calculate factorial value of n numbers?

1!= 1

2!= 1x 2=2

3!=1x2x3=6

4!=1x2x3x4=24

5!=1x2x3x4x5=120

n!=1x2x3x4x5......xn

create function fact(@n int)

returns int

as

begin

declare @i int,@f int

set @i=1

set @f=1

while @i<=@n

begin

set @f=@f\*@i

set @i=@i+1

end

return @f

end

select dbo.fact1(5)

--create a function to display yesterday date

create function yesterday()

returns datetime

as

begin

declare @d1 datetime

select @d1=getdate()-1

return @d1

end

select dbo.yesterday()

--create a function to display nextday

create function nextday()

returns varchar(20)

as

begin

declare @day varchar(20)

set @day=datename(dw,getdate()+1)

return @day

end

select dbo.nextday()

create function months\_between(@d1 datetime,@d2 datetime)

returns int

as

begin

declare @diff int

set @diff=datediff(mm,@d1,@d2)

return @diff

end

select dbo.months\_between(getdate(),'06-18-2016')

--Write a tsql program to get the years, months and days between two dates.

Declare @dateofbirth datetime

Declare @currentdatetime datetime

Declare @years varchar(40)

Declare @months varchar(30)

Declare @days varchar(30)

set @dateofbirth='1988-12-06'--birthdate

set @currentdatetime =getdate()--current datetime

select @years=datediff(year,@dateofbirth,@currentdatetime)-- To find Years

select @months=datediff(month,@dateofbirth,@currentdatetime)-(datediff(year,@dateofbirth,@currentdatetime)\*12)

-- To Find Months

select @days= datepart(d,@dateofbirth) - datepart(d,@currentdatetime)-- To Find Days

select @years +' years, ' +@months +' months, '+@days +' days' asYearMonthDay

3.How to convert starting letter of each word to capital? using sql server

ex: hello hai how are you

output: Hello Hai How Are You

create FUNCTION InitCap ( @Input varchar(40) )

RETURNS VARCHAR(40)

AS

BEGIN

DECLARE @I INT

DECLARE @Char CHAR(1)

DECLARE @PrevChar CHAR(1)

DECLARE @Output VARCHAR(25)

SET @Output = LOWER(@Input)

SET @I = 1

WHILE @I <= LEN(@Input)

BEGIN

SET @Char = SUBSTRING(@Input, @I, 1)

SET @PrevChar = CASE WHEN @I = 1 THEN ' '

ELSE SUBSTRING(@Input, @I - 1, 1)

END

IF @PrevChar =' '

SET @Output = STUFF(@Output, @I, 1, UPPER(@Char))

SET @I = @I + 1

END

RETURN @Output

END

select dbo.InitCap('hello hai how r u')

4.create a function that takes countryname as an argument and returns capital name?

CREATE FUNCTION Capital (@Country varchar(15))

RETURNS varchar(30)

AS BEGIN

declare @return varchar(30)

select @return = case @Country

when 'Argentina' then 'South America'

when 'Belgium' then 'Europe'

when 'Brazil' then 'South America'

when 'Canada' then 'North America'

when 'Denmark' then 'Europe'

when 'Finland' then 'Europe'

when 'France' then 'Europe'

when 'US' then 'Washington'

when 'India' then 'NewDelhi'

else 'Unknown' end

return @return

end

--how to invoke

select dbo.Capital('Finland')

5.create FUNCTION DecimalToBinary(@a int)

RETURNS varchar(20)

AS

BEGIN

DECLARE @b varchar(20) =' '

WHILE @a > 0

BEGIN

SET @b= @b + CAST((@a % 2) AS varchar)

SET @a = @a / 2

END

RETURN REVERSE(@b)

END

select dbo.decimaltobinary(20)

6.Create a function to Convert lower case to uppercase after a full stop/period

Create FUNCTION Test1(@val varchar(40))

RETURNS varchar(40)

AS

BEGIN

DECLARE @len int, @i int

DECLARE @res varchar(40)

SET @res = ''

SET @len = LEN(@val)

SET @i=1

WHILE(@i <= @len)

BEGIN

IF @i=1

BEGIN

SET @res = @res + UPPER(SUBSTRING(@val,@i,1))

END

ELSE

BEGIN

IF SUBSTRING(@val,@i,1) = '.' AND SUBSTRING(@val,@i+1,1) <> ' '

BEGIN

SET @res = @res + SUBSTRING(@val,@i,1) + UPPER(SUBSTRING(@val, @i+1,1))

SET @i = @i+1

END

ELSE IF SUBSTRING(@val,@i,1) = '.' AND SUBSTRING(@val,@i+1,1) = ' '

BEGIN

SET @res = @res + SUBSTRING(@val,@i,2) + UPPER(SUBSTRING(@val, @i+2,1))

SET @i = @i+2

END

ELSE

BEGIN

SET @res = @res + SUBSTRING(@val,@i,1)

END

END

SET @i=@i+1

END

RETURN @res

END

select dbo.test1('hello how are u. i am fine. time is gold')

--display ename in the following format

smith,jones,sekar,james.....

create procedure display

as

begin

declare @ename varchar(20)

declare @s varchar(400)=' '

declare c1 cursor for select ename from emp

open c1

fetch next from c1 into @ename

while @@FETCH\_STATUS=0

begin

set @s=@s+@ename+','

fetch next from c1 into @ename

end

print @s

close c1

deallocate c1

end

exec display

II) TABLE VALUED FUNCTIONS :

\* These functions will return multiple rows.

\* When these functions are created Begin and End blocks should not be mentioned.

\* A function should be provided with return type as TABLE, since it is returning multiple rows.

Syntax:

CREATE FUNCTION func\_name(list\_of\_args)

RETURNS TABLE

AS

RETURN (SELECT query)

Calling a Table Valued Functions:

syntax:

SELECT \* FROM dbo.func\_name(list\_of\_args)

example:

1.create a function called empret that retrieves the rows of those employees who are working in deptno 20?

CREATE FUNCTION empret(@dno INT)

RETURNS TABLE

AS

RETURN (SELECT \* FROM emp WHERE deptno=@dno)

SELECT \* FROM dbo.empret(20)

select \* from dbo.empret(10)

2.Write a function to return entire dept table?

create function dept\_table()

returns table

as

return(select \* from dept)

--how to reterive the data from multiple tables?

create function deptemp()

returns table

return(select emp.ename,dept.dname from emp inner join dept on

emp.deptno=dept.deptno)

select \* from deptemp()

calling table valued function :

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

select \* from dbo.dept\_table()

To display all the user created functions:

SELECT NAME FROM SYSOBJECTS WHERE XTYPE='FN'

--To display all table valued function

select name from sysobjects where xtype='IF'

viewing the code related to stored functions:

sp\_helptext func\_name : this predefined procedure will display the complete code of a function which is created in a databas.

sp\_helptext fact

Droping a function:

DROP FUNCTION func\_name

DROP FUNCTION sumnos1

Stroed Block:

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

1.Stored procedure -- exec p1

2.stored function -- select dbo.fun1();

3.stored trigger or database trigger -- automatic calling

common table expression:

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

The recursive CTE structure must contain at least one anchor member and one recursive member. The following pseudocode shows the components of a simple recursive CTE that contains a single anchor member and single recursive member.

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

AS

(

CTE\_query\_definition –- Anchor member is defined.

UNION ALL

CTE\_query\_definition –- Recursive member is defined referencing cte\_name.

)

-- Statement using the CTE

SELECT \*

FROM cte\_name

The semantics of the recursive execution is as follows:

Split the CTE expression into anchor and recursive members.

Run the anchor member(s) creating the first invocation or base result set (T0).

Run the recursive member(s) with Ti as an input and Ti+1 as an output.

Repeat step 3 until an empty set is returned.

Return the result set. This is a UNION ALL of T0 to Tn.

example :

with xx(ename,eno)

as

(select ename,eno from EMP1)

select \* from xx

USE production

GO

-- Define the CTE expression name and column list.

WITH Sales\_CTE (empno,ename,doj)

AS

-- Define the CTE query.

(

SELECT empno,ename,YEAR(hiredate) doj FROM emp

)

-- Define the outer query referencing the CTE name.

SELECT empno, COUNT(empno) AS Total, doj

FROM Sales\_CTE

GROUP BY doj,empno

ORDER BY doj,empno

GO

Database Trigger or Stored Trigger

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

\* Database trigger is a stored procedure that is fired when an insert,update or delete statement is issued against the associated table.

uses:

------

1.To generate data automatically.

2.To enforce complex integrity constraints(checking with sysdate,checking with data in another table)

3.To customize complex security authorization.

4.To maintain replicate tables.

5.To audit data modification.

bank\_master

accno balance

1001 10000

1002 20000

1003 20500

1004 40000

1005 55555

bank\_transaction

cname accno ttype amount

xxx 1003 db 2500 exec p1

yyy 1005 cr 20000

yyy 1005 db 10000

two types of trigger:

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

1.table level trigger (only one time firing) => after

2.row level trigger ( 100 row deleted => 100 time firing) => for

3.view =instead of

syntax:

create/alter trigger <trigger\_name>

on <tablename> / <viewname>

after/instead of/for insert,update,delete

as

begin

statements

end

Practial step:

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

step1 : Create bank\_master table with sample data

create table bank\_master(accno int,balance int)

insert into bank\_master values(1001,10000),(1002,20000),(1003,30000),(1004,40000),(1005,50000)

step2 : create bank\_transaction empty table

create table bank\_transaction(cname varchar(20),accno int,ttype char(2),amount int)

step3:

CREATE TRIGGER T11 ON BANK\_TRANSACTION AFTER INSERT

AS

BEGIN

DECLARE C1 CURSOR FOR SELECT \* FROM BANK\_TRANSACTION

DECLARE @CNAME VARCHAR(20),@ACCNO INT,@TTYPE VARCHAR(2),@AMOUNT INT

OPEN C1

FETCH NEXT FROM C1 INTO @CNAME,@ACCNO,@TTYPE,@AMOUNT --START THE LOOP

WHILE(@@FETCH\_STATUS=0)

BEGIN

FETCH NEXT FROM C1 INTO @CNAME,@ACCNO,@TTYPE,@AMOUNT -- CONTINUE THE LOOP

END

IF @TTYPE='CR'

UPDATE bank\_MASTER SET BALANCE=BALANCE + @AMOUNT WHERE ACCNO=@ACCNO

ELSE

UPDATE bank\_MASTER SET BALANCE=BALANCE - @AMOUNT WHERE ACCNO=@ACCNO

CLOSE C1

DEALLOCATE C1

END

2.Create a trigger tr2 For INSERT,DELETE Triggering

event where trigger should be fired if the transaction

is performed before 10AM and After 4PM?

CREATE TRIGGER tr2

ON bank\_master

FOR INSERT,DELETE

AS

BEGIN

IF DATEPART(HH,GETDATE()) not in(10,11,12,13,14,15,16)

BEGIN

ROLLBACK

RAISERROR('INVALID TIME',1,1)

END

END

1- Error Number

1 - Error state

3. create a trigger tr3 for INSERT,DELETE triggering

event where trigger should be fired if the transactions are performed on SUNDAY?

CREATE TRIGGER tr3

ON bank\_master

FOR INSERT,DELETE

AS

BEGIN

IF DATENAME(DW,GETDATE())='Sunday'

BEGIN

ROLLBACK

RAISERROR('CANT INSERT OR DELETE THE DATA ON SUNDAY',1,1)

END

END

Saturday (10 am to 2pm)

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

4. create a trigger tr1 for INSERT,DELETE triggering

event where trigger should be fired if the transactions are performed on 2nd and 4th Saturday before 10am and after 5pm ?

alter TRIGGER tr3

ON bank\_master

FOR INSERT,DELETE

AS

BEGIN

declare @counter int=0,@stdate datetime,@enddate datetime

set @stdate=getdate() - day(getdate()-1)

set @enddate=getdate()

while @stdate <=@enddate

begin

if datename(dw,@stdate) in('Saturday') --day of week

set @counter=@counter+1

set @stdate=@stdate +1

end

IF DATENAME(DW,GETDATE())='Saturday' and @counter in(2,4)

BEGIN

ROLLBACK

RAISERROR('CANT INSERT OR DELETE THE DATA ON SATURDAY',1,1)

END

END

insert into bank\_master values(1006,60000)

Assignment:

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

5. create a trigger tr1 for INSERT,DELETE triggering

event where trigger should be fired if the transactions are performed on 2nd and 4th Saturday.

5.Create a trigger t12 For INSERT Triggering

event where trigger should be fired if the input amount is more than balance.

alter TRIGGER T11 ON BANK\_TRANSACTION AFTER INSERT

AS

BEGIN

SET NOCOUNT ON --STOP THE SERVER MSG

DECLARE C1 CURSOR FOR SELECT \* FROM BANK\_TRANSACTION

DECLARE @CNAME VARCHAR(20),@ACCNO INT,@TTYPE VARCHAR(2),@AMOUNT INT,@BALANCE INT

OPEN C1

FETCH NEXT FROM C1 INTO @CNAME,@ACCNO,@TTYPE,@AMOUNT --START THE LOOP

WHILE(@@FETCH\_STATUS=0)

BEGIN

FETCH NEXT FROM C1 INTO @CNAME,@ACCNO,@TTYPE,@AMOUNT -- CONTINUE THE LOOP

END

SELECT @BALANCE=BALANCE FROM BANK\_MASTER WHERE ACCNO=@ACCNO

IF @AMOUNT > @BALANCE and @TTYPE='DB'

BEGIN

ROLLBACK

RAISERROR('INVALID AMUONT',1,1)

END

ELSE IF @TTYPE='CR'

UPDATE bank\_MASTER SET BALANCE=BALANCE + @AMOUNT WHERE ACCNO=@ACCNO

ELSE

UPDATE bank\_MASTER SET BALANCE=BALANCE - @AMOUNT WHERE ACCNO=@ACCNO

CLOSE C1

DEALLOCATE C1

END

--how to know howmany triggers are created?

select name from sysobjects where xtype='tr'

--how to view the trigger code?

sp\_helptext 't11'

--how to drop the trigger?

drop trigger t11

--Write a trigger to halt the transaction of the employees SALESMAN & PRESIDENT?

Create trigger t11

on emp for insert,update,delete

as

begin

declare @job varchar(20)

select @job=job from inserted where job in ('salesman','President')

select @job=job from deleted where job in('salesman','president')

if (@job is not null)

begin

rollback

raiserror('Invalid operation',1,1)

end

end

Difference between procedures and Triggers:

procedures Triggers

1.called Explicitly (Exec p1) 1.called implicitly

2.can be created without a table. 2.can't be created without a table.

3.it accepts parameters 3.it won't accept parameters

4.it returns a value(output) 4.it won't return a value.

VIRTUAL TABLES (MAGIC TABLES)

SQL SERVER provides 2 virtual tables which can be used only in triggers.

These tables are provided by TempDB(System database).

These tables play major role to have backup of data.

Tables are Identified as INSERTED , DELETED

Insert - INSERTED

delete - DELETED

update -delete + insert

INSERTED :

This table will store the same data

which is provided in target table(user defined table).

If a record is inserted into a target table

then the same record is available in this virtual table.

Data avaiable in this table can be used to

perform operations and can be again stored into the other

user defined tables.

If the record is updated in a target table,

new value is stored in this INSERTED table and old value is transfered to DELETED Table.

DELETED :

This magic table stores the removed rows from a target table.

It also stores the old value when update operation

is performed over a target table.

note:

These tables will store the data for temporary.

These tables can be used at once in a single trigger.

6.CREATE a trigger tr4 for DELETE triggering event on

DEPT table,where trigger should be fired by deleting

the records from emp table?

CREATE TRIGGER TR7

ON DEPT

AFTER DELETE

AS

BEGIN

SET NOCOUNT ON --STOP THE SERVER MSG

DELETE FROM EMP WHERE DEPTNO IN(SELECT DEPTNO FROM DELETED)

PRINT CAST(@@ROWCOUNT AS VARCHAR(5))+ 'ROWS ARE DELETED'

END

Table - After - Table level

Table - for - row level

View - Instead of - Table level

INSTEADOF TRIGGERS:

These triggers is used to make the modifications

into base table thru a complex view.(Non updateable view)

By default a complex view is not updatable view (i.e. read only view).

A complex view consists of joins,mathematical expressions, group by clause, group functions, distinct operator.

example:

create a complex view on emp table that stores a query for empno,sal and annual salary?

CREATE VIEW V1

AS

SELECT EMPNO,SAL M\_SAL,SAL\*12 A\_SAL FROM EMP

CREATE TRIGGER tr6

ON V1

INSTEAD OF INSERT

AS

BEGIN

INSERT EMP(EMPNO,SAL) SELECT EMPNO,M\_SAL FROM INSERTED

end

create a database trigger do not allow to enter more than four employees in a emp4 table.

create trigger tr1 on

emp4

for insert

as

begin

declare @cnt int

declare @ename varchar(20),@eno int,@salary int,@deptno int

select @ename=ename,@eno=eno,@salary=salary,@deptno=deptno from

inserted

select @cnt=count(\*) from emp4 where deptno=@deptno

if @cnt >=4

raiserror('max 4 employees',15,1)

else

insert into emp4 values(@ename,@eno,@salary,@deptno)

end

select \* from emp4

--SQLSERVER 2008 ON DDL TRIGGER (CREATE,ALTER,TRUNCATE,DROP)

\* Create a trigger for DDL triggering event that restricts the droping and altering of a table in a database?

CREATE TRIGGER tr5

ON DATABASE

FOR DROP\_TABLE,ALTER\_TABLE,CREATE\_TABLE

AS

BEGIN

ROLLBACK --cancel the transaction

RAISERROR('CANT ALTER OR DROP THE TABLES',15,16)

END

Nested Trigger:

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

Trigger Recursion and Nesting

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

\* Trigger events can be fired within another trigger action.

\* One Trigger execution can trigger event on another table or same table.

\* This trigger is called NESTED TRIGGER or RECURSIVE TRIGGER.

\* Nested triggers SQL Server supports the nesting of triggers up to a maximum of 32 levels.

\* Nesting means that when a trigger is fired, it will also cause another trigger to be fired.

create table t1(a int)

create table t2(a int)

create trigger t1it on t1

for insert as

print 'begin tr11 firing'

insert t2 values(1) --Nested trigger

print 'end tr11'

go

create trigger t2it on t2 for insert as

print 'begin t2it'

print 'end t2it'

insert t1 values (1) --manually

select \* from t1

select \* from t2

TSQL - DML After Trigger for Insert,Update and Delete Operation

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

Scenario:

Sometime we have to write a trigger on table to capture changes for different operation such as Insert, Update and Delete. Here is sample Code that can be modified according to the Source Table.

Solution:

I have created dbo.Customer Table as Source Table on which I want to create DML After trigger. After that I have created an Audit Table for dbo.Customer with name dbo.Customer\_Audit which is going to save all changes.

CREATE TABLE Customer

(

CustomerID INT IDENTITY(1, 1),

Name VARCHAR(50),

ADDRESS VARCHAR(100)

)

GO

CREATE TABLE Customer\_Audit

(

[CustomerID] INT,

[Name] VARCHAR(50),

[ADDRESS] VARCHAR(100),

[OperationDate] [DATETIME] NOT NULL,

[Operation] [VARCHAR](50) NOT NULL,

[OperationBy] [VARCHAR](100) NOT NULL

)

--Create DML After Trigger

Create TRIGGER Tr12 ON Customer FOR INSERT, UPDATE, DELETE

AS

SET NOCOUNT ON;

--Capture the Operation (Inserted, Deleted Or Updated)

DECLARE @operation AS VARCHAR(10)

DECLARE @Count AS INT

SET @operation = 'Inserted'

SELECT @Count = COUNT(\*) FROM DELETED

IF @Count > 0

BEGIN

SET @operation = 'Deleted'

SELECT @Count = COUNT(\*) FROM INSERTED

IF @Count > 0

SET @operation = 'Updated'

END

--Capturing Delete Operation

IF @operation = 'Deleted'

BEGIN

INSERT INTO dbo.Customer\_Audit

([CustomerID],

[Name],

[ADDRESS],

[OperationDate],

[Operation],

[OperationBy])

SELECT [CustomerID],

[Name],

[ADDRESS],

GETDATE() AS [OperationDate],

'Deleted' AS [Operation],

Suser\_name() AS [OperationBy]

FROM deleted

END

ELSE

BEGIN

--Capturing Insert Operation

IF @operation = 'Inserted'

BEGIN

INSERT INTO dbo.Customer\_Audit

([CustomerID],

[Name],

[ADDRESS],

[OperationDate],

[Operation],

[OperationBy])

SELECT [CustomerID],

[Name],

[ADDRESS],

GETDATE() AS [OperationDate],

'Inserted' AS [Operation],

Suser\_name() AS [OperationBy]

FROM inserted

END

--Capture Update Operation

ELSE

BEGIN

INSERT INTO dbo.Customer\_Audit

([CustomerID],

[Name],

[ADDRESS],

[OperationDate],

[Operation],

[OperationBy])

SELECT [CustomerID],

[Name],

[ADDRESS],

GETDATE() AS [OperationDate],

'Updated' AS [Operation],

Suser\_name() AS [OperationBy]

FROM inserted

END

END

To display list of triggers created in a database:

\* SELECT NAME FROM SYSOBJECTS WHERE XTYPE='TR'

\* select \* from sys.all\_objects where type='tr'

VIEWING TRIGGER INFORMATION:

SP\_HELPTEXT trigger\_name

SP\_HELPTEXT tr2

DISABLE / ENABLE THE TRIGGERS:

ALTER TABLE table\_name DISABLE/ENABLE TRIGGER trigger\_name

--how to disable or enable db level trigger?

DISABLE / ENABLE TRIGGER all ON database

DROPING A TRIGGER:

DROP TRIGGER trigger\_name

SECURITY

SQL SERVER 2014 provides a connection to a database server in 2 ways:

1. Windows Authentication : At this authentication a user connects to database server using windows accounts i.e. users existing at windows operating system. It is called trusted connection.

2. SQL SERVER Authentication : At this authentication a user connects to database server using the SQL SERVER accounts. By default SQL SERVER provides a login called sa, for which blank password cant be provided in SQL SERVER 2005, whereas in previous versions blank password for sa can be created.

by default login : sa

password : \*\*\*\*\*\*\*\*\*\*\*\*\*

CREATING A NEW LOGIN:

To create a new login,

it is essential that user should enter into a database

which has got system administration privileges.

Syntax:

CREATE LOGIN <login\_name> WITH PASSWORD='passwd'

DEFAULT\_DATABASE='db name'

CHANGING THE PASSWORD:

To change the password of existing login, the following command is used in that database which has got Administration privileges.

Syntax:

ALTER LOGIN <login\_name> WITH PASSWORD='passwd'

DROPING A LOGIN:

To drop a login user should be in a database

which has got administration privileges.

Syntax:

DROP LOGIN <login\_name>

Examples:

login name : Ram (serverlevel)

Database : Ashok

user : Ram (Database level)

Table : emp,dept,salgrade

role : ram\_role

emp : select

dept: select,insert

salgrade : select,insert,update

To create a login:

USE MASTER

CREATE LOGIN GUNA WITH PASSWORD='123' DEFAULT\_DATABASE='Batch2'

To Change the password:

USE MASTER

ALTER LOGIN GUNA WITH PASSWORD='123'

To Drop the Login:

USE MASTER

DROP LOGIN GUNA

CREATING A LOGIN WITH GUI:

From view menu click on Object explorer--->

Databases-->Security--->

Right click on Logins--->click on new login

Login Name : Rambabu

Select SQL SERVER Authentication

Password : 123

Confirm Password : 123

Deselect Enforce Password Ploicy

Default Database : Practice

From SELECT A PAGE window,

select User Mapping, and Select the database,

where in that user it creates a user(Peers)

Click OK.

Changing the password and properties of Login:

From View menu click on object explorer--->

Databases---->Security--->Logins--->

Select the login and right click properties

Change the necessary options and then click ok.

SQL SERVER AGENT :

In SQL Server 2012 SQL Server Agent is a service.

It supports to create Jobs, Schedules, Alerts And Operators.

job: job is nothing but a task.

schedule : Timing schedule

alert is nothing but a predefined condition.

operator is nothing but a responsible person who will receive the notification.

It provides automatic execution of jobs and tasks associated with different steps based on system date and time.

To work with SQL Server Agent it is essential that services should get start.

--Example1:

-

--Every 1 minutes to insert the records into emp1 table automatically

step1 : Create the table

create table emp1(eno int identity,doj datetime)

--EXAMPLE2 :

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

\* EVERYDAY 5PM TO TAKE THE syed DATABASE BACKUP AUTOMATICALLY.

backup database syed to disk='c:\maulik\syed.bak'

--how to run the job manually?

Right click on the jobname -> start job at step

--how to monitor the job?

Right click on the jobname -> viewhistory

--how to monitor all the jobs?

click on -> job Activity monitor

--how to stop the jobs? (enable or disable)

right click on the job name -> disable / enable

To start the service the following steps should be used.

start--->programs--->Microsoft SQL Server 2005---> Configuration Tools--->SQL Server Configuration Manager---> Select SQL Server Agent (double click on the services and click on start)

To work with SQL Server Agent;

creating a table:

CREATE TABLE EMP1

(ENO INT IDENTITY,DOJ SMALLDATETIME)

Object Explorer

|

Expand SQL Server Agent

Creating a job, Steps and Schedules:

STEP 1:

Select the jobs, right click and new job

Name : ins\_rec\_job

Owner : sa

Category: [Uncategorized [lOCAL]]

STEP 2:

select "steps" from "select a page window"

Click on New

Step Name : ins\_rec

Type : Transact-SQL script (T-SQL)

Database : praveen

Command : INSERT EMP1 VALUES(getdate())

Click Ok

STEP 3:

Select "Schedules" from "Select a page window"

Click New

Name : sch\_ins\_rec

Schedule Type : Recurring

Frequency

Occurs : Daily

Recurrs Every : 1 Day

Daily Frequency

Occurs Every : 1 Min

Click Ok

Click Ok

Example2:

----------

\*EVERY DAY 5PM TO TAKE AUTOMATIC BACKUP OF YIHUI DATABASE

BACKUP COMMAND SYNTAX:

BACKUP DATABASE YIHUI TO DISK='C:\YIHUI.BAK'

--HOW TO RUN THE JOB MANUALLY?

--HOW TO VERIFY JOB?

--HOW TO ENABLE OR DISABLE THE JOB?

--HOW TO VERIFY ALL JOBS?

NOTE:

To execute the created procedure automatically based on a created schedule.

CREATE PROCEDURE insrow

AS

BEGIN

INSERT EMP1 VALUES(getdate())

END

Under steps window, create a new step and provide the command used to execute the procedure;

EXEC insrow

SP\_RENAMEDB: Here SP stands for Stored Procedure. This stored procedure is used to change the name of the existing database.

Syntax: SP\_RENAMEDB ‘OLD DATABASENAME’, ‘NEW DATABASENAME’

E.g.: SP\_RENAMEDB ‘NRSTT’, ‘NRSTTS’

The above statement renames (changes the database name) NRSTT to NRSTTS

SP\_RENAME: This stored procedure is used for changing the name of the table and for changing the name of the column

i. Syntax to change the name of the table

SP\_RENAME ‘OLD TABLENAME’, ‘NEW TABLENAME’

E.g. SP\_RENAME ‘EMP’, ‘EMPLOY’

The above stored procedure changes the name of EMP table to EMPLOY

ii. Syntax to change the name of the column

SP\_RENAME ‘TABLE.OLDCOLUMN NAME’, ‘NEW COLUMNNAME’

E.g. SP\_RENAME ‘STUDENT.ADR’, ‘ADDRESS’

The above stored procedure changes the name of ADR column to ADDRESS in STUDENT table.

SP\_HELP: This stored procedure is used to display the description of a specific table.

Syntax: SP\_HELP TABLENAME

E.g.: SP\_HELP EMP

The above stored procedure displays the description of EMP table

SP\_DATADASES: This Stored procedure displays the list of databases available in SQL Server.

Syntax: SP\_DATABASES

SP\_TABLES: This stored procedure displays the list of tables available in the current database.

Syntax: SP\_TABLES

SP\_HELPDB: This stored procedure is used to display the description of master and log data file information of a specific database

Syntax: SP\_HELPDB Database-Name

Ex: SP\_HELPDB SAMPLE

SP\_SPACEUSED: This stored procedure is used to find the memory status of the current database

--how to identify user connection created in database?

SELECT

DB\_NAME(dbid) as DBName,

COUNT(dbid) as NumberOfConnections,

loginame as LoginName

FROM

sys.sysprocesses

WHERE

dbid =37

GROUP BY

dbid, loginame

--Write a trigger to halt the transaction of the employees SALESMAN & PRESIDENT?

Create trigger t11

on emp for insert,update,delete

as

begin

declare @job varchar(20)

select @job=job from inserted where job in ('salesman','President')

select @job=job from deleted where job in('salesman','president')

if (@job is not null)

begin

rollback

raiserror('Invalid operation',1,1)

end

end

Difference between procedures and Triggers:

procedures Triggers

1.called Explicitly (Exec p1) 1.called implicitly

2.can be created without a table. 2.can't be created without a table.

3.it accepts parameters 3.it won't accept parameters

4.it returns a value(output) 4.it won't return a value.

VIRTUAL TABLES (MAGIC TABLES)

SQL SERVER provides 2 virtual tables which can be used only in triggers.

These tables are provided by TempDB(System database).

These tables play major role to have backup of data.

Tables are Identified as INSERTED , DELETED

Insert - INSERTED

delete - DELETED

update -delete + insert

INSERTED :

This table will store the same data

which is provided in target table(user defined table).

If a record is inserted into a target table

then the same record is available in this virtual table.

Data avaiable in this table can be used to

perform operations and can be again stored into the other

user defined tables.

If the record is updated in a target table,

new value is stored in this INSERTED table and old value is transfered to DELETED Table.

DELETED :

This magic table stores the removed rows from a target table.

It also stores the old value when update operation

is performed over a target table.

note:

These tables will store the data for temporary.

These tables can be used at once in a single trigger.

6.CREATE a trigger tr4 for DELETE triggering event on

DEPT table,where trigger should be fired by deleting

the records from emp table?

CREATE TRIGGER TR7

ON DEPT

AFTER DELETE

AS

BEGIN

SET NOCOUNT ON --STOP THE SERVER MSG

DELETE FROM EMP WHERE DEPTNO IN(SELECT DEPTNO FROM DELETED)

PRINT CAST(@@ROWCOUNT AS VARCHAR(5))+ 'ROWS ARE DELETED'

END

Table - After - Table level

Table - for - row level

View - Instead of - Table level

INSTEADOF TRIGGERS:

These triggers is used to make the modifications

into base table thru a complex view.(Non updateable view)

By default a complex view is not updatable view (i.e. read only view).

A complex view consists of joins,mathematical expressions, group by clause, group functions, distinct operator.

example:

create a complex view on emp table that stores a query for empno,sal and annual salary?

CREATE VIEW V1

AS

SELECT EMPNO,SAL M\_SAL,SAL\*12 A\_SAL FROM EMP

CREATE TRIGGER tr6

ON V1

INSTEAD OF INSERT

AS

BEGIN

INSERT EMP(EMPNO,SAL) SELECT EMPNO,M\_SAL FROM INSERTED

end

create a database trigger do not allow to enter more than four employees in a emp4 table.

create trigger tr1 on

emp

for insert

as

begin

declare @cnt int

declare @ename varchar(20),@eno int,@salary int,@deptno int

select @ename=ename,@eno=eno,@salary=salary,@deptno=deptno from

inserted

select @cnt=count(\*) from emp4 where deptno=@deptno

if @cnt >=4

raiserror('max 4 employees',15,1)

else

insert into emp4 values(@ename,@eno,@salary,@deptno)

end

select \* from emp4

--SQLSERVER 2008 ON DDL TRIGGER (CREATE,ALTER,DROP)

\* Create a trigger for DDL triggering event that restricts the droping and altering of a table in a database?

CREATE TRIGGER tr5

ON DATABASE

FOR DROP\_TABLE,ALTER\_TABLE,CREATE\_TABLE

AS

BEGIN

ROLLBACK --cancel the transaction

RAISERROR('CANT ALTER OR DROP THE TABLES',15,16)

END

Nested Trigger:

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

Trigger Recursion and Nesting

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

\* Trigger events can be fired within another trigger action.

\* One Trigger execution can trigger event on another table or same table.

\* This trigger is called NESTED TRIGGER or RECURSIVE TRIGGER.

\* Nested triggers SQL Server supports the nesting of triggers up to a maximum of 32 levels.

\* Nesting means that when a trigger is fired, it will also cause another trigger to be fired.

create table t1(a int)

create table t2(a int)

create trigger t1it on t1

for insert as

print 'begin tr11 firing'

insert t2 values(1) --Nested trigger

print 'end tr11'

go

create trigger t2it on t2 for insert as

print 'begin t2it'

print 'end t2it'

insert t1 values (1) --manually

select \* from t1

select \* from t2

TSQL - DML After Trigger for Insert,Update and Delete Operation

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

Scenario:

Sometime we have to write a trigger on table to capture changes for different operation such as Insert, Update and Delete. Here is sample Code that can be modified according to the Source Table.

Solution:

I have created dbo.Customer Table as Source Table on which I want to create DML After trigger. After that I have created an Audit Table for dbo.Customer with name dbo.Customer\_Audit which is going to save all changes.

CREATE TABLE Customer

(

CustomerID INT IDENTITY(1, 1),

Name VARCHAR(50),

ADDRESS VARCHAR(100)

)

GO

CREATE TABLE Customer\_Audit

(

[CustomerID] INT,

[Name] VARCHAR(50),

[ADDRESS] VARCHAR(100),

[OperationDate] [DATETIME] NOT NULL,

[Operation] [VARCHAR](50) NOT NULL,

[OperationBy] [VARCHAR](100) NOT NULL

)

--Create DML After Trigger

Create TRIGGER Tr12 ON Customer FOR INSERT, UPDATE, DELETE

AS

SET NOCOUNT ON;

--Capture the Operation (Inserted, Deleted Or Updated)

DECLARE @operation AS VARCHAR(10)

DECLARE @Count AS INT

SELECT @Count = COUNT(\*) FROM DELETED

set @operation='Inserted'

IF @Count > 0

BEGIN

SET @operation = 'Deleted'

SELECT @Count = COUNT(\*) FROM INSERTED

IF @Count > 0

SET @operation = 'Updated'

END

--Capturing Delete Operation

IF @operation = 'Deleted'

BEGIN

INSERT INTO dbo.Customer\_Audit

([CustomerID],

[Name],

[ADDRESS],

[OperationDate],

[Operation],

[OperationBy])

SELECT [CustomerID],

[Name],

[ADDRESS],

GETDATE() AS [OperationDate],

'Deleted' AS [Operation],

Suser\_name() AS [OperationBy]

FROM deleted

END

ELSE

BEGIN

--Capturing Insert Operation

IF @operation = 'Inserted'

BEGIN

INSERT INTO dbo.Customer\_Audit

([CustomerID],

[Name],

[ADDRESS],

[OperationDate],

[Operation],

[OperationBy])

SELECT [CustomerID],

[Name],

[ADDRESS],

GETDATE() AS [OperationDate],

'Inserted' AS [Operation],

Suser\_name() AS [OperationBy]

FROM inserted

END

--Capture Update Operation

ELSE

BEGIN

INSERT INTO dbo.Customer\_Audit

([CustomerID],

[Name],

[ADDRESS],

[OperationDate],

[Operation],

[OperationBy])

SELECT [CustomerID],

[Name],

[ADDRESS],

GETDATE() AS [OperationDate],

'Updated' AS [Operation],

Suser\_name() AS [OperationBy]

FROM inserted

END

END

To display list of triggers created in a database:

\* SELECT NAME FROM SYSOBJECTS WHERE XTYPE='TR'

\* select \* from sys.all\_objects where type='tr'

VIEWING TRIGGER INFORMATION:

SP\_HELPTEXT trigger\_name

SP\_HELPTEXT tr2

DISABLE / ENABLE THE TRIGGERS:

ALTER TABLE table\_name DISABLE/ENABLE TRIGGER trigger\_name

--how to disable or enable db level trigger?

DISABLE / ENABLE TRIGGER all ON database

--table level

disable trigger all on emp

DROPING A TRIGGER:

DROP TRIGGER trigger\_name