JUNE BR

3/06/2013 MON

SOL

Rs=110/=

Data : Collection of facts and figures

→ 1. STORING DATA IN BOOKS AND FILES:

Disadvantages:

1) Costly in maintanance
2) Required lots of man power
3) Not secure
4) Retriving data will be time consuming.

when Computers are comes to world, there is no any Software to store the file and that time all these Files are stored in Flat files (or) Text Files. For which we can reduce the manpower. So automatically production cost is decreased and profit will be increased.

→ 2. FLAT FILES OR TEXT FILES IN COMPUTER:

Drawbacks:

1) Retriving of data is complicated because we need to write different application programs for retriving.
2) Security problems comes into picture while managing the data
3) Data Redundancy and in-consistancy due to multiple copies of the data
4) Data Integrity (maintaing proper data) can be adoped because we can't impose any business rules on the data Present under a text file.
5) No indexing mechanism for accessing the data from text files.

DATABASE: It is a specialized software for managing the data than storing the data.

4-Jun-19 TUE

DATA STORE OR DATA SOURCE:

It is a location where we can store the data, where data is a collection of fact and figures.
we have different types of data stores available like,

-> Books and files
-> Flat files on a computer
-> Database.

DATABASE: A database is a special software that is purely designed for managing the data.

In between 60's & 70's the concept of databases was born mainly to resolve the problems or drawback we are facing with the older system like books, flat files etc.

Databases have given a permanent solution for the various problems we have been facing earlier like,

DATA RETRIVAL: It is a process of retriving information from a datastore. which is very complex while retriving data from flat file which was addressed with a special language.

SEQUEL: [Structured English Query Language]
-That can retrive a language from a database as simple as possible.

SECURITY: Data is never secure under books and flat file where as database are provided an excellent "ROLE BASE SECURITY MECHANISM" For accessing the data to secure the data which includes a process of authonication on authorization.

DATA REDUNDANCY AND DATA IN-CONSISTANCY.

These problems comes into picture when we maintain multiple copies of the data where the changes are made in one copy will not be reflect to other copy in case of books and flat file but in case of database we can maintain No. of. copies of the same data and still the changes made a one copy reflects to the other.

DATA INTEGRITY: This is about maintaining proper data and to maintain proper data every organization imposes a set of rules on the data and we call these rules as business Rule. Database provides an option of imposing the business rules with the help of Constraints and triggers.

INDEXING: Indexes are used for accessing the data much more faster where a flat file doesn't provide any proper indexing mechanism but data bases users an excellent indexing mechanism to access the ~~data~~ from any where in the sameway.

Databases are of two types
-> OLTP (on Line Transaction processing)
->OLAP (ConLine Analytical processing)

The OLTP is For managing the current on day to day information where as OLAP is to maintain the past or history of the data. Which is mainly used for data Analysis and can also be called as WAREHOUSES.

DBMS: [DATA BASE MANAGEMENT SYSTEM]

It is a subject which tell how to organize the data in a more efficient way. Basing on the subject there are various data base model which came into existance time to time like,

HDBMS [Hierarchial Database management system] one-Many
NDBMS [Network Database management system] Many-Many
RDBMS [Relational Database management system]
ORDMS [object Relational Database management system]
OORDBMS [object oriented Relational Database Management system]

5-June-13 Wed

HDBMS [Hierarchial DataBase Management system]

This is the first DBMS model that came into existance in the 60's which will organize the data in the form of a tree structure or organizational structure.

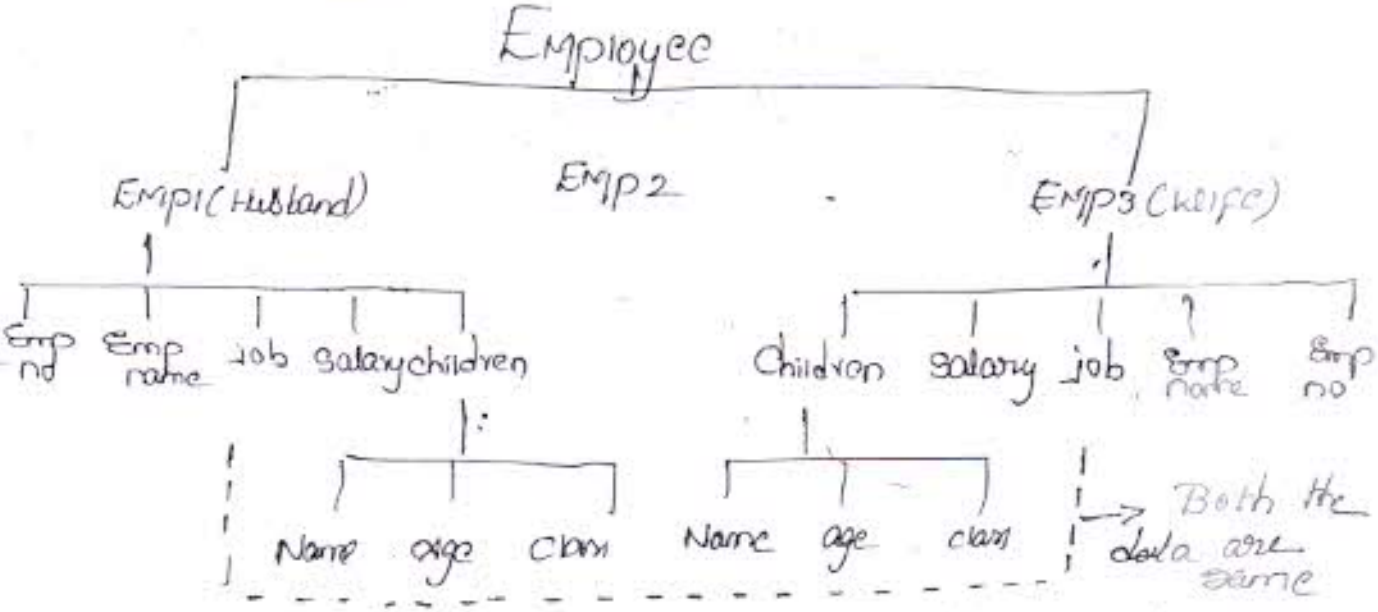
To represent the data in DBMS we first identify an entity. i.e Whose data has to be managed.

Ex:- customer, Employee, student etc.

Anything which has attributes to get can called as entity. Once we identify the entities we can identify the of that entity and put the data in a hierarchical order as following

Employee:-

→ Empno
→ Empname
→ Job
→ Salary
→ children.



HDBMS is designed on the relationship model one to many.

Drawbacks of HDBMS:-

→ Because it is designed in one to many relationship a child can have only a single parent so, redundancy comes into Picture because of data duplication.

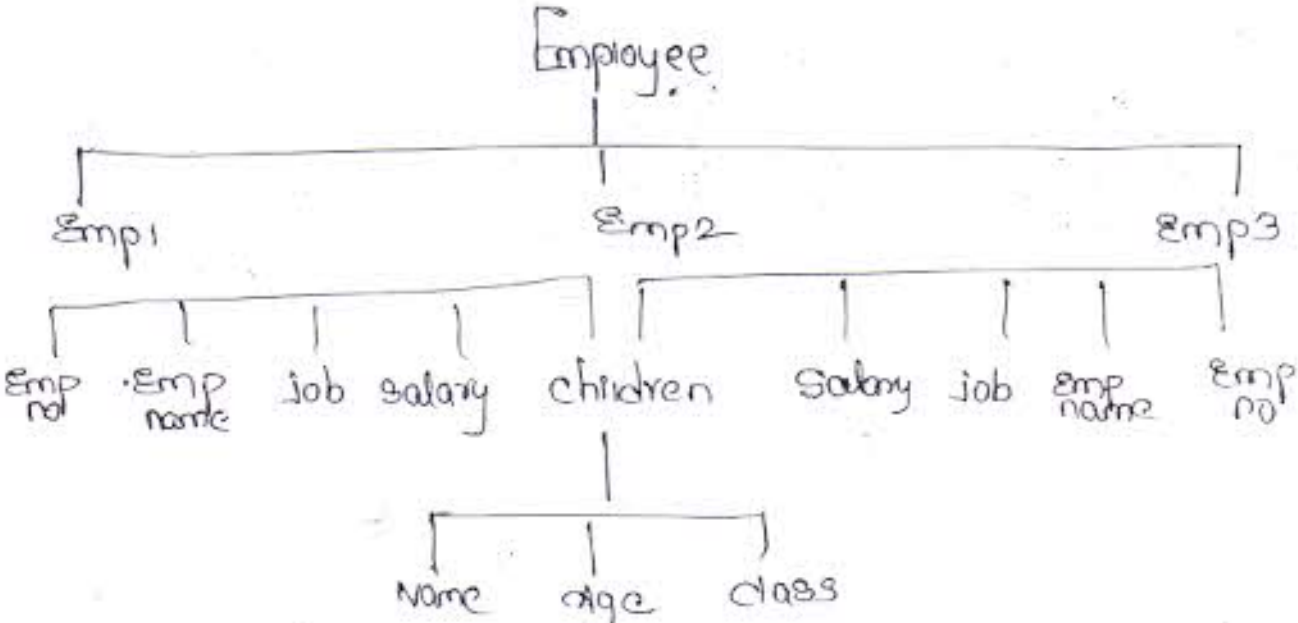
→ When we want to access the data in this model we need to travel from the root node step by step to the desired level, which will be time consuming.

The first database that came into existance in a hierarchical model is IMS [Information, Management System] from IBM

Currently the windows registry under windows os runs in the hierarchical model only.

NDBMS: [Network Database Management System]

The Network model is a modification to the existing hierarchical model bringing many to many in to the picture so that a child can have multiple parent which can eliminate the redendancy.



→ Accessing the data in this model is very faster because it uses pointers for identification of the data

→ In 1969 the first NDBMS was launched with the Name as IDBMS [Integrated Database Management System]

→ There are no major drawbacks in this model but still the immediate take over by Relational model in 70's didnot make the Network model successful.

RDBMS: [Relational Database Management System]

In HDBMS and NDBMS data is organised in the form of a tree structure which looks complex to manage and understand also. So to overcome this problem in 70's Dr. E.F. Codd from IBM came with a new concept on storing the data in a table structure i.e Rows and columns of data.

⇒ E.F. Codd with all his ideas for the new model called as Relational model has published an article with the title as

"A Relational Model of Data for Large Shared Data Bank"

-> Basing on this above article many companies came forward like IBM, Relational Software INC [currently oracle] etc. has started the designed for the new database model i.e RDBMS.

->RDBMS is mainy based on two mathematical principles Relational Algebra and calculus.

-> In the year 70s IBM has given the prototype for RDBMS known as "System R"

-> In the year 1974 IBM has launched a language for communication with RDBMS known as SEQUEL.

-> The first Relational database was launched in the industry in the year 1978 with the name multi relational data store followed by Berkley Ingress QUEL and IBM Bi12

-> In the year 1979 Relational software INC has launched the most popular database in the market right now i.e oracle.

6-06-2019 THRUSDAY

CODD RULES:

Dr.E.F codd in his RDBMS specifications has proposed a set of roles that has to be satisified by database to be called as Relational and we call those rule as Codd Rules.

There are 12 codd Rules which are proposed by codd and any database will be called as Relational needs to satisfy atleast six of the available Rules.

1. THE INFORMATION RULE: According to this data under a Relational database must be stored in the form of a table that is Data in the form of rows and columns.

2. THE GUARANTEED ACCESS RULE:

According to this every row of the table must have a unique identification for accessing the row. so, that we can easily pick the desired row from the table. To do this there should be a column in the table which doesn't contain (or) allow duplicates and NULL values and we call that column as a identity column.

To maintain a column as identity we can take the support of primary key constraint.

3. SYSTEMATIC TREATMENT OF NULL VALUES RULE:

NULL
The value is unknown

According to this a database should allow storing of NULL values under a column of a table provided the value is unknown and this NULL value should be capable of being stored under any datatyped column like a string or int or float or boolean

4. THE VIEW UPDATING RULE:

A view is a logical table which is used for maintaining multiple copies of the data without Redundancy. The best thing in a view is any modification we perform on the table Reflects into the view.

According to the view updating rule the views which are displaying the data should also be updatable and those changes must be Reflected back into the table.

SEQUEL (Structured English Query Language)

Its a language that has been designed for communication with any RDBMS and it is common for all RDBMS products.

SEQUEL now known as SQL is controlled by ANSI (American National standard Institution) which gives the specifications for SQL that has to be followed and

implemented by all the database vendors (owners) (whoever the own databases) like MICROSOFT, ORACLE, IBM

SQL is a declarative programming Language where as programming languages like C, C++, Java, C# comes under the category of imperative programming languages.

SQL:

SQL is further divided into Five subpoints Like

DDL (Data Definition Language)
DML (Data manipulation Language)
DQL (Data query Language)
DCL (Data control Language)
TCL (Transaction control language)

DDL deals with the structure of the data. It provides Four commands on it like,

- Create, Alter, drop and Truncate

DML deals with the data directly and it contains three commands on it like,

- Insert, update and delete

DQL also deals with the data only but for retrieving the data we use it which contains only a single command

- SELECT

DCL deals with the security of the data (or manage) by setting the permissions for users to access (or) restrict the data access. This contains three commands in it like,

- Grant, Revoke and Deny

TCL deals with Transaction management which contains the three command in it Like,

- Commit, Rollback and Savepoint.

(Here the two statements should execute at the same time or none of the statement should execute) Eg:- Transfering a amount from one account to other account.

(SQL Server) database but SQL is a Language

7/06/2013 FRIDAY

SQL SERVER: It is also an RDBMS that has been designed and developed by adopting the codd rules

The development of SQL server database has been done as following

* In 1987 Sybase released SQL Server for unix operating system.
* In 1988 Microsoft, Sybase and Aston Tate joined hands to launch SQL Server For OS/2.

Note:- OS/2 is an operating system that is designed by Microsoft and IBM.

* In 1989 Microsoft, Sybase and Aston Tate released SQL Server 1.0 For OS/2.
* In 1990 SQL server 1.1 is released with support for windows 3.0 Client-s.
* In the same year Aston Tate drops out of SQL Server development.
* In 1991 Microsoft and IBM ended joint development of OS/2.
* In 1993 Microsoft launched windows NT 3.1
* In the same year Microsoft and Sybase released Version 4.2 of SQL Server for windows NT.
* In 1994 Microsoft and Sybase ceased development of SQL-Server officially ended and microsoft continued to develop the windows version of SQL server and Sybase continued to develop the Unix version of SQL server

UNIX Version

* In 1995 Microsoft released version 6.0 of SQLServer
* In 1996 Microsoft released version 6.5 of SQLServer
* In 1998 Microsoft released version 7.0 of SQLServer

In 2000 Microsoft released SQL Server 2000
In 2005 Microsoft released SQL server 2005 on November 2005
In 2008 Microsoft released SQL server 2008 on August 2008
In 2010 Microsoft released SQL Server 2008 R2 - on April 2010.
In 2012 Microsoft released SQL Server 2012 on on March 2012

Version	year	ReleaseName	Codename
1.0(OS/2)	1989	SQLServer1.0	-
1.1(OS/2)	1991	SQLServer 1.1	-
4.21(WinNT)	1993	SQLServer4.2.1	SQLNT
6.0	1995	SQLServer 6.0	SQL95
6.5	1996	SQLServer 6.5	Hydra
7.0	1998	SQLServer 7.0	Sphinx
8.0	2000	SQLServer 2000	Shiloh
9.0	2005	SQLServer 2005	Yukon
10.0	2008	SQLServer 2008	Katmai
10.5	2010	SQLServer 2008R2	Kilimanjaro
11.0	2012	SQL Server 2012	Denali

Note:-
To work with SQL Server we have two database type
1. database Engine (or) Server
2. database client

→ The database Engine is a server where by using this server we can work on different system.
→ database Client is the one where the client use in the system (user)
Eg:- ATM.

To work on SQL server we use
SQL Server Management Studio.
It is a tool used for the SQL Server.

If we connect to Server It shows a window with

Server type

Server name

Server Authentication

Login

Password.

In the Sever type We have 4 types

→ Database Engine
→ Integrated Server
→ Analysis Server } OLAP
→ Reporting Server

But we prefer Database Engine, because it cannot be seen in the operating system. We work on it.

Server name should have the user name

Sever type- we have two type
→ Windows Authenication
→ SQL Server Authenication.

Windows Authenication works on the user admin and it is secured one. and here it has the own database Engine. database client.

SQL Server Authenication works on the current user Password when it is installed we have to, give that Password

But on the operating system if we give default it takes Windows Authentication,
But if we mixed Form we will have both Windows & SQL server authentication.

Admin Part in the operating System.

SAT
8/06/2019

Working With SQLServer:

While working with SQL server first we need to choose a version of SQLserver we want to work with. After choosing the version we want to work with it we need to choose the Edition under this version.

Where Microsoft makes SQLserver available in multiple editions with different features sets and targetting different users.

Currently we are provided with the following editions under SQL server

1. Data center:- It is a fully featured Edition designed for data centers that need high levels of application support. These Edition has been retired in SQLserver 2012.

2. Enterprise edition:- This Edition includes the code database engine and add on services, with a range of tools for creating and managing a SQL server cluster. It can manage databases as large as 524 petabytes and address 2 tetrabytes of memory and also support 8 physical processor before 2012 and from 2012 it is 160 physical processors

3. Standard:- This Edition include the code database Engine along with the standalone services. It differs from Enterprise Edition in that it supports fewer active instances and does not include some high availability functions such as 'hot-add memory (allowing memory to be added while the server is running) and Parallel indexing.

4. Web:- This Edition is an option available for web hosting

5. Business Intelligence:- This Edition is introduced in SQLserver 2012 focusing on Self Services and corporate business intelligence.

Work Group – This Edition includes the core database functionality but does not include any additional services. This Edition has been retired in SQL Server 2012.

Express – This Edition is a scaled down, free Edition of SQL Server which includes only the core database engine. Two additional Editions provides a superset of features not in the original Express Edition. The first is SQL Server Express with Tools which includes SQL Server Management Studio Basic. The second is SQL Server Express with advanced services which acts full-text search capabalities and reporting services.

Express
- Sql server Express – database engine
- Sql server Express with Tools – database Engine, management Studio
- Sql server Express with servites – database Engine, management Studio and Reporting services

When we install SQL server on our machine it will be installed as two seperate components are parts that is Every database software has those two parts.

→ Database Engine (or) Server
→ Database client.

Database Engine is the place where we store the data and it is integerated with the OS which can be found as an operating system services.

Note:- After installing SQL server to test the server running under the OS control go to start menu and type "Services.msc" under the search text box. Which displays a list of operating system services which are running in the back ground process In that list we can find the database engine running.

(SQL Server (MSSQLSERVER) 14 Provides to Running Automatic)

Database Client - SQL Server provides a database client tool which acts as a interface to communicate with the database server that is "SQL server Management Studio". Whenever we open the management studio we need to first connect with the data Engine con Server to perform any database operations.

To Launch management studio. Go to
→ Start menu select All programs
↓
Microsoft SQL Server
↓
double click on SQL Server management studio.

once the management studio opens it displays a window known as connect to Server using which we need to connect with the database Engine. under dat window we need to supply the following details

1. Server type – Choose Database Engine in it
2. Server Name - specify the name of the machine on which the database Engine or Server is running. If it is on the local machine specify the name of local machine only or simply Enter Local (or) Dot (.)
3. Authentication - It is a processor of verfying the credentials of a user to login into the server. SQL server supports two different Authentication model for login into the server.
   1. Windows Authentication
   2. SQL Server Authentication

While installing an SQL Server we must select an Authentication mode for database Engine. Either Windows Authentication mode (default) or Mixed mode

If we select Windows Authentication when the user connects to the server it validates the account name & password using the windows principal token in the operating System. So SQL server does not ask for any password and doesnot perform any identity validation.

When the user connects through the SQL server using SQL server Authentication Logins are created in SQL server that are not based on window user account Both the user name and Password are created by using SQL server and stored in SQLserver So users connecting with this Authentication mode must provide their credentials every that they connect.

* Note:- Windows Authentication is a default mode of and it is much more secured than SQL server Authentication. because it uses "Kerberos Security Protocol" for enforcing Password Security. (it is a real time environment used for securing)

If using SQL server Authentication, we need to supply the login name and password under connect to server and click on the connect button where as if we choose windows (enabled) Authentication directly click on the connect button (disabled) which connects to the sever and we can start performing operations on the Server.

11/06/2013 - Tuesday (Page No-22)

Working With Sql Server:

Sql Server is a collection of databases where a database is again a collection of various objects like Tables, Views procedures, functions, Triggers, Indexes ..etc

Databases under into sql server are divided into 2 categories

1. System databases
2. User databases.

1. System databases: These databases are used by sql for the functionality of sql server and without these databases the server will not function. users of sql server are not given a direct support for updating the information under system databases. The system databases include these four databases

   1. Master
   2. Model
   3. MSdB
   4. TempdB

1. Master: These databases records all the system level information under associated with the server

2. Model: It is used as a template for all new databases created on sql server database

   i.e Whenever we create a new database a copy of the model database is taken and given with a new name.

3. MSDB :- It is used for recording or storing information about alerts, Jobs etc. that are performed by SQL Server agent
eg: (backup files)

4. TempDB :- It is used as a workspace for holding the temporary tables that are created under various databases. once we restart the server the temp db database is destroyed and again recreated. So, we will not find any temporary tables that are created previously.

User databases :- These are databases created and managed by the users for storing their objects like tables views ellra.

To create a database open the management studio and in the left hand side, we find the option called window object Explorer, open it and right click on the node Databases and select new database which opens a window asking for database name. Enter a name under it.

Ex:- Sql11 and click OK.

Which creates a databases and adds it under the databases node

Creating a database using create statement

we can also create a database by using Create Command which should be used as following

Syntax:- Create Database <dbname>

<> : Anyvalue
[] : optional

→ To create a table in the above process, in the management Studio, we will find a button New query under the toolbar, click on it which opens a window, under it write the following statement

Create Database Sqlll

→ To execute the statement, again in the toolbar we find a button, execute click on it which will parse and execute the statement.

Whenever a database created on sqlserver, it will have two operating system files

1. A datafile
2. A Logfile

DATAFILE: This is the file which contains data and objects such as a tables, indexes, stored procedures etc. and this data files can either be a primary or secondary datafile

Primary datafile contains the start-up information for the database and pointers to other files associated with the database. (ndf)

Every database can have only one primary datafile and Saved with an Extension .mdf (masterdatafile)

secondary datafile are optional and user defined, we use

Secondary datafile, when the data is spread across multiple disks by putting each file on a different disk drive. We can have any number of secondary datafiles and they are with a extension .ndf (next datafile).

LOGFILES: These files contain all the information that is required to recover all transactions in the database, because it holds transactional log information. There must be atleast one log file for each database which is saved with an extension .ldf (Log datafile).

To view the mdf and ldf files of our database

Sql11 (C:\ old drive)

→ Go to <drive>:\program Files\Microsoft sql Server\MSSQL11.MSSQLSERVER\MSSQL\Data

Sql11.mdf
Sql11_log.ldf

Wed
12/06/2013

Datatypes in SQL server:-

A datatype is an attribute that specifies what type of data a column can hold in it like integer data or character data or monetary data or date and time data or binary strings, and so on.

SQL server supplies a set of system datatypes that defined all the type of data that can be stored within SQL server. Data types in SQL server are organized into following categories.

1. Exact numerics
2. Approximate numeric
3. Date and time
4. Character strings
5. Unicode character strings
6. binary strings
7. other data types

INTEGER TYPES: Exact Number datatypes that stores integer data.

Datatype	Range	Storage
Bigint	-2^{63} to $2^{63}-1$	8 bytes
int	-2^{31} to $2^{31}-1$	4 bytes
Smallint	-2^{15} to $2^{15}-1$	2 bytes
tinyint	0 to 255	1 byte

MONEFARY OR CURRENCY TYPES: Datatypes dat. represents monetary or currency values

Datatype	Range	Storage
Money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8bytes
Smoll Money	-214,748.3648 to 214,748.3647	4bytes

Bit (BOOLEAN TYPE): An integer type that can take a value of 1, 0 or NULL.

Note:- The string values TRUE AND FALSE converted to Bit values. TRUE is converted to 1 and false is converted to 0. Converting to bit promotes any nonzero value to 1.

DECIMAL TYPES:

Syntax:-

decimal [(P[,S])] and numeric [(P[,S])]

Numeric data types that have fixed precision and scale.

P(precision): The maximum total number of decimal digits that can be stored, both to the left and to the right of decimal point. The precision must be a value from 1 through the maximum precision of 38. The default precision is 18.

S(scale): The maximum number of decimal digits that can be stored to right of the decimal point. Scale must be a value from 0 through p. scale can be specified only if precision is specified. The default scale is 0. Maximum storage sizes vary based on the precision, as following

Precision	Storage bytes
1-9	5
10-19	9
20-28	13
29-38	17

Float [(n)] and Real: Approximate number datatypes for use with floating point numeric data. Floating point is approximate. Therefore not values in the datatype can be represented exactly. Where n is the no. of bits that are used to store the mantissa of the float number in scientific notation, and, therefore dictates the precision and storage size. If n is specified it must be a value b/w 1 and 53. The default value of n is 53.

n value	Precision	Storage size
1-24	7 digits	4 bytes
25-53	15 digits	8 bytes

Date and TIME Types: SQL Server supports following date and time types.

1. Date: Defines a date
2. Time [(fractional seconds_precision)]: Defines a time of a day. The time is without time zone awareness and is based on a 24 hour clock.
3. DateTime: Defines a date that is combined with a time of day with fractional seconds. that is based on 24-hour clock.
4. SmallDateTime: Defines a date that is combined with a time of day. The time is based on a 24-hour day, with seconds always zero (:00) and without fractional seconds.

5. Datetime2 [(Fractional seconds precision)]: Defines a date that is combined with a time of day that is based on 24 hour clock. DateTime2 can be considered as an Extension of the above DateTime type that has a larger data range.

6. DateTimeOffset: [(Fractional seconds precision)]: Defines a date that is combined with a time of a day that has time zone awareness and is based on 24-hour clock.

Datatype	Range	Storage
Date	January 1, 1 A.D. to December 31, 9999 A.D.	3 bytes
Time	00:00:00:0000000 to 23:59:59:9999999	5bytes
DateTime	Date Range: January 1, 1753, to December 31, 9999 Time Range: 00:00:00 to 23:59:59:997	8bytes.
Small DateTime	Date Range: January 1, 1900, to June 6, 2079 Time Range: 00:00:00 to 23:59:59	4bytes
Date-time 2	Date Range: January 1, 0001 AD to December 31, 9999 AD	6 bytes if Precision < 3. 7 bytes if Precision is 3 & 4.

	Time Range: 00:00:00 to 23:59:59.9999999	8 bytes for other precisions
DateTimeOffset	Date Range: January 1, 0001 AD. to December 31, 9999 AD Time Range: 00:00:00 to 23:59:59.9999999	10 bytes

CHARACTER TYPES: Are string datatypes of either fixed length or variable length

1. char[(n)]: Fixed length, non-unicode string data, n defines the string length and must be a value from 1 to 8,000. The storage size is n bytes.

2. Varchar[(n|max)]: Variable length, non-unicode string data, n defines the string length and can be a value from 1 to 8,000. Max indicates that the maximum storage size is $2^{31}-1$ bytes (2 GiB). The storage size is actual length of the data entered + 2 bytes.

3. text: Same as varchar(max).

4. nchar[(n)]: Fixed length unicode string data. n defines the string length and must be a value from 1 to 4000. The storage size is two times of n bytes.

5. nvarchar[(n|max)]: Variable length unicode string data. n defines the string length and can be a value from 1 to 4000. Max indicates that the maximum storage size is

$2^{31}-1$ bytes (2GB). The storage size in bytes is two times the actual length of data entered + 2 bytes.

6. ntext: Same as nvarchar(max).

BINARY TYPES: Binary datatypes of either fixed length or variable length for storing binary values like images, Audio and videos.

1. binary [(n)]: fixed length binary data with a length of n bytes where n is a value from 1 to 8000. The storage size is n bytes.

2. varbinary [(n/max)]: Variable length binary data. n can be a value from 1 to 8,000. max indicates that the maximum storage size is $2^{31}-1$ bytes (2GB). The storage size is the actual length of the data entered + 2 bytes

3. image: Same as varbinary(max).

NOTE:- text, ntext and image datatypes will be removed in the future version of Microsoft SQL Server. Avoid using these datatypes in new development work, and plan to modify applications that currently use them. use nvarchar(max), varchar(max), and varbinary(max) instead.

n - represents - National (nvarchar)

13/06/2013 Thursday

OTHER DATA TYPES:

1. ROWVERSION: Is a data type that exposes automatically generated unique binary numbers within a database. Row version is generally used as a mechanism for version stamping table rows. The storage size is 8 bytes. The row version datatype is an incrementing number and does not preserve a date or a time.

* NOTE:- upto 2005, of SQL SERVER we have a datatype Timestamp which is used for version stamping table rows. That datatype is deprecated in the current version and introduced with row version. So, it is not advised to use timestamp anymore.

2. UNIQUE IDENTIFIER: Is a 16 byte GUIT (Global unique identifier) a column of this datatype can be initialized with a value by using the newId function.

3. XML :- Is a datatype that stores xml data we can store instances of wellformed xml in a column of xml type. The storage representation of xml datatype instances cannot exceed 2GB in size.

4. SQL_Variant :- A column of this type can store values of different datatypes. For example a column defined as sql variant can store int, binary and character values also. It can have a maximum length 8016 bytes. This includes both a base type information and the base type value. The maximum length of the actual base type value is 8000 bytes.

5. Cursor: A datatype for variables or stored procedure o/p parameters that contain a reference to a cursor. The cursor datatypes cannot be used for a column in a CREATE TABLE Statement.

6. Table: Is a special datatype that can be used to store a result set for processing at a later time in a function or stored procedure.

Creating a table:-

Syntax:- Create Table <tnames> (<colname><dtype>[,...n])

→ Table Name must be unique under the database
→ The name of the table should be ranging b/w 1 to 128 characters
→ Never start table names with numerics (or) special characters and also do not use spaces in table names. If there are multiple words combine them with an (_).
→ The maximum No. of columns a table can have in the current version of SQL Server is 1024.

Ex:- To create a table for customers in bank

```
Create table (ustomer (custid int, Cname varchar(50),
Balance Decimal (9,2), DOB SmallDatetime, city
varchar(50))
```

To create the table under a database first open Management studio provide the Authentication details and login. Now click on the New Query button on the top which opens a query window on the top of it we find a combo box listing all the databases

Choose your database (Sqyll) and then click on the create button which creates the table in the selected database

After creating the table if we want to know the structure of the table any time use the statement

Syntax:- SP_ Help <tname>

Eg:- SP_ Help Customer

INSERTING RECORDS INTO THE TABLE:

Syntax:- Insert into <tname> [(<collist>)] values (<vallist>)

RETRIEVING DATA FROM A TABLE:

Syntax:- Select * [collist] From <tname> [<clauses>]

- Insert into customer values (1001, 'Scott', 5000, '06/19/2013', 'Hyderabad')

String and Date values has to be enclosed under single quotes

While entering date we need to give the date in either mm/dd/yyyy or yyyy/mm/dd formats.

Ex:

- Select * From customer.

Inserting Values into Selected Columns

Insert into customer (custid, cname, Balance) values (1002, 'Smith', 5000).

Note:- If a value is not inserted into a column it will take the default value as NULL. Because the default default value for a column is NULL.

We can write the above statement like this also,

Insert into customer (1003, 'Blake', 7000, NULL, NULL)

Note: While using NULL in the insert statement do not put it under Single Quote.

Inserting values into the table when we are aware of the columns but not aware of column sequence

Insert into customer (city, cname, custid, AOD, Balance) values ('Delhi', 'James', 1004, '2013/06/13', 10000)

using ~~Get Date~~ GetDate() for inserting values into Date column

Insert into customer values (1005, 'williams', 16000, GetDate(), 'Chennai')

→ GetDate() is a predefined function which returns the server date and time.

Retriving all Records of the data:

Select * From customer

Retriving Selected Columns:

Select custid, cname, Balance From customer

Retriving columns of a table in a desired order:-

Select custid, cname, city, AOD, Balance From customer

Retriving columns of a name with alias name:-

Select custid, cname as [customer Name], city, AOD as [Account opening Date], Balance From customer.

Note:- Alias should be represented in [ ] where we provide spaces.

Retrieving Selected records from table:-

Select * From customer Where Balance > 5000

Note:- Where clause is used for restricting unwanted records from the table.

Select * From customer Where city IS NULL

While comparing a column value with NULL we should not use Equal (=) and not Equal (≠) operators in that we should use IS NULL and IS NOT NULL. Because Null is a infinite or indefinite or unknown value. So, no two null values can be compared with each other.

14-June-2013

Updating (update command):-

update command is used for modifying in a existing data that is present under a table.

Syntax:- update <tname> Set <column> = <value> [,...n]
[<Where condition>].

Ex:- update customer Set Balance = 6000 Where custid = 1001

Note:- While updating or deleting a record a Where condition is mandatory for unique identification of the record.

Multiple columns can be update

Ex:- update customer Set Balance = 4000, city = 'Hyderabad' where custid = 1002.

Ex:- update customer set AOD = GetDate() Where AOD IS NULL

AOD
NULL
NULL

- DELETE COMMAND:- It is used for deleting record(s) From a table

Syntax:- Delete from <tname> [Where condition>]

Ex:- Delete From customer Where Cutid = 1005

Droping a table:- It is a process of destroying the Existence of the object in the database.

Syntax:- Drop table <tname>

Ex:- Drop table tcutomey

DATA INTEGRITY: In computing, data integrity refers to maintaining and assuring the accuracy and consistency of the data over its entire Life-cycle and is an important feature of database (or) RDBMS system. Data integrity means that the data contained in the database is accurate and reliable.

To provide integrity of data RDBMS provides us a set of integrity constraints which ensures that data Entered into the database is accurate valid and consistent.

Integrity constraints are basically divided into three types of Those are

→ 1. Entity Integrity, not allowing multiple rows to have same identity within a table.

Eg:- Primary key & unique.

→ 2. Domain Integrity: Restricts a data to predefined data types.
Eg:- Dates., Eg:- check

→ 3. Referential Integrity, Required the existence of a related row in another table.
Eg:- Foreign key.

* CONSTRAINTS :- SQL Server Supports Five constraints for maintaining data integrity those are
1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. CHECK
5. FOREGINKEY

* Note:- constraints are imposed on columns of a table.

1. NOT NULL :- If these constraint is imposed on a column, that column will not allow null values into it.

Syntax:- Create Table <tname (<coname> <dtype> Not Null
<coname> <dtype> NOTNULL[, ...])

Eg:- Create Table customer (
custid int Not NULL, cname varchar(50), Balance Decimal(9,2)
Addr smalldatetime Not NULL, city varchar(50))

* Note:- NOT NULL constraints can be imposed on any number of columns.

Ex:- Insert into customer values(100, 'Raju', 5000, NULL, 'Hyderabad')

In the above case we are trying to insert a column value AOD on which the Not null constraint is imposed so, the insert statements execution fails.

Note: When we insert a null value into a column on which the Not null constraint is imposed, the execution of the insert statement is terminated by displaying a user friendly error message telling the reason for termination and also specifies the database, the table and the column where the problem got occured. (Error).

UNIQUE CONSTRAINT: If this constraint is imposed on a column, that column will not allow duplicate values into it.

Syntax:- Create Table <tname>(
<colname> <dtype> [constraint <const name>]
<const type> [,...n])

Note:- Here constraint type in the sense, it can be unique (or) primary key (or) check (or) foregin key also.

Ex:- Create table customer (custid int unique, cname varchar(50), Balance Decimal(9,2), AOD smalldatetime NOT NULL, city varchar(50))

Note:-
If we insert a duplicate value into a column that contains unique constraint on it, the statements get terminated by displaying an error message but that error message will not contain the column name where the violation got occured. It only displays the constraint name

∴ unique, primary key, check and foregin key constraints

are independent objects under the database which are Created and linked with the column of the table. So, they have their own identity or name i.e, the reason why if these four constraints are violated they will never ben the column where the violation occured. They will only display their name.

Actually a constraint name must be defined by us while creating the table. If not defined by us database Server will give a name to that constraint and those names will not be user friendly for identification. So, it is always suggested to provide our own name to the constraint for Easy identification by adopting a naming convention as following.

<Columnname>_<constraint type>

Ex:- creating a table by giving own name to a constraint

```
Create table customer (
Custid int constraint custid_unique unique,
Cname varchar(50), Balance decimal (9,2), AGD
Smalldatetime NOT NULL, city varchar(50));
```

Monday.
17/06/2013

IMPOSING CONSTRAINTS ON TABLE:

We can impose constraints on a table in two different ways:

1. Column level imposing a constraint
2. Table level imposing a constraint

In the first case, we provide the constraint information beside the column only where as in the second case we first define all the columns and then we define constraints on the columns.

* NOTE:- We cannot impose not null constraint in Table level. It is possible only for rest of the four constraints.

Column level imposing a constraint

Create table customer (custid int constraint custid_unique unique, cname varchar(50), Balance decimal(9,2), Aod smalldatetime NOT NULL, city varchar(50));

Table level imposing a constraints

create table customer (
custid int, cname varchar(50), Balance decimal (9,2),
Aod smalldatetime NOT NULL, city varchar(50),
constraint custid_unique unique (custid));

Composite Constraint:-

There is no difference in behaviour whether the constraint is imposed in table level (or) column level but if a constraint is imposed in table level we have an advantage of imposing composite table level constraints. That is one constraint on multiple columns.

Branch Details

city	Branch code	Location BranchName
Hyderabad	B_1	Ameerpet
Hyderabad	B_2	Punjagutta
Hyderabad	B_3	Jubilee Hills
Mumbai	B_1	Dadar
Mumbai	B_2	Powai
Mumbai	B_3	Thane
Chennai	B_1	Gopalapuram
Chennai	B_2	Ayanavaram
Chennai	B_3	Moniyamapuram
Delhi	B_1	Lajpat nagar
Delhi	B_2	Nilad colony
Delhi	B_3	connaught place

In the above table city and Branchcode columns must be imposed with a composite constraint.

Eg:- Create table BranchDetails (
City varchar(50), Branchcode varchar(10), Branch Location varchar(50),
Constraint city_Bc_UQ unique (City, Branch code)).

*NOTE:-

The drawback with a not NULL constraint is it will allow duplicate values where as incase of unique constraints it will allow null values.

PRIMARY KEY CONSTRAINT: It is a combination of unique and Not NULL which doesn't allow NULL as well as duplicate values into a column (or) columns. Using primary key we can enforce entity integrity. We can impose only one primary key constraint on a table which can be either on a single or multiple columns also. i.e compaite Primary key is allowed.

Ex:- Column level imposing primary-key constraint

```
Create Table Customer (
custid int Constraint custid_PK primary key,
Cname varchar(50), Balance Decimal(9,2), MOD smalldatetime NOT NULL, City varchar(50));
```

Table level imposing primary key constraint

```
Create Table BranchDetails (
Qty varchar(50), Branchcode Varchar(10), Branch location varchar(50),
Constraint City_BC_PK primary key (City, Branchcode)
)
```

CHECK CONSTRAINT: It is a constraint which enforces domain Integrity by limiting the possible values that can be entered into a columns (or) column.

Eg:- Column level imposing constraints

```
Create table Customer(
custid INT constraint custid_PK primaryKey, cname
varchar(50), Balance Decimal(9,2) Constraint Balance_
CK check( Balance >= 1000), CAOD SmallDateTime NOT NULL,
City varchar(50));
```

Table level imposing constraints

```
create table Customer( custid int, cname varchar(50),
Balance Decimal (9, 2), CAOD SmallDatetime Not NULL,
City varchar(50),
Constraint custid_PK primary key (custid),
Constraint Balance_ck check (Balance >= 1000));
```

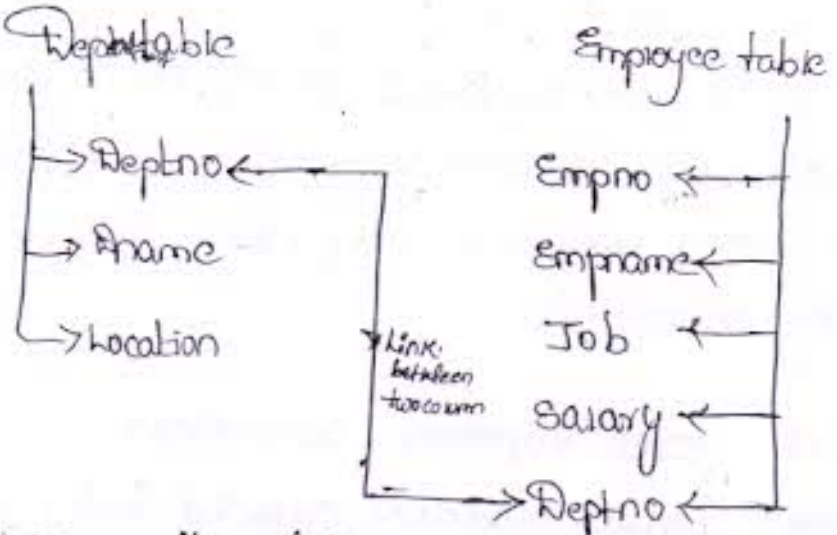
TUE
18/06/2013

FOREIGN KEY CONSTRAINT: This constraint is used for Providing referential integrity for the data that is present in a column (or) columns that is it requires the existence of a related row in another table. To impose a foreign key constraint we require the following things

NOTE: Foreign key constraint is used for relating (or) binding two tables with each other and then verify the existence of one tables data in the other.

We require two tables for binding with each other and those two tables must have a common column for linking the tables.

Eg:-



In the above case, the dept no. column of Dept table is being referred under the Employee table. So in this context Department table is referred as a master or a parent table and the Employee table is referred as Detailed table (or) child table.

Note:- The common column that is present in both the two tables need not have the same name but their datatypes must be same.

The common column that is present under the parent or master table is known as a reference key column. And reference key column should not contain duplicate values in it. So, we need to impose either unique (or) primary key constraint on that column.

If we impose primary key constraint on the reference key column that column will also be the identity column of the table

The common column which is present in the child or detailed table is known as Foreign key column. And we need to impose a foreign key constraint on the column which refers to reference key column of Master table.

Department table → (Master (or) parent table)

Reference Key column.

Deptno	Dname	Loc
10	Sales	Mumbai
20	Production	Chennai
30	Finance	Delhi
40	Research	Hyderabad

Employee → (Detail (or) child table)

Empno	Ename	Job	Salary	Deptno → Foreign Key cantnn
1000				10
1001				10
1002				10
1003				10
1004				20
1005				20
1006				20
1007				20
1008				30
1009				30
1010				30
1011				30
1012				40

To create department table

Create Table Department (Deptno int constraint Deptno_PK primary key, Dname varchar(50), Loc varchar(50));

Insert values

```
Insert into Department values (10, 'Sales', 'Mumbai');
Insert into Department values (20, 'Production', 'Chennai');
Insert into Department values (30, 'Finance', 'Delhi');
Insert into Department values (40, 'Research', 'Hyd..');
```

To create child table:

Imposing Foreign key constraint in column level:

```
Create table Employee (Empno int, Ename varchar(50), Job
Varchar(50), Salary Decimal(7,2), Deptno Int constraint
Deptno FK References Department (Deptno))
```

Impaing Foreign key constraint in Table level:

```
Create table Employee (Empno int, ename varchar(50), Job
Varchar(50), Salary Decimal(7,2), Deptno int,
Constraint Deptno_FK Foreign key (Deptno) references
Department (Deptno))
```

NOTE: While imposing a foreign key constraint in table level we need to explicitly use the foreign key clause to specify the foreign key column where as it is not required while imposing in column level because the constraint is defined beside the foreign key column only.

When we impose the foreign key constraint and establish relations between the table, the following three rules will come into picture.

RULE1 :- Cannot insert a value into the foreign key column Provided that value is not existing under the reference key column of master table.

RULE2 :- Cannot delete a record from the master table Provided that records reference key value has child records in the child table without addressing what to do with the child records.

RULE3 :- cannot update the reference key value of a master table provided that value has corresponding child records in the detail table without addressing what to do with the child records.

Wed 19/06/13

If we want to delete (or) update a record in the master table when they have corresponding child records in the child table we are provided with a set of rules to perform delete and update operations known as Delete rules and update rules.

DELETE RULES:

On Delete No action
On Delete cascade
On Delete Set NULL
On Delete Set Default

Update Rules

on update No Action
on update cascade
on update Set NULL
on update Set Default

ON DELETE NO ACTION (or) ON UPDATE NO ACTION:

This is the default rule that is applied on the child tables that restrict deleting a record in the master table (or) updating a reference key of a master table when corresponding child records are present under the child table.

ON DELETE CASCADE (&) ON UPDATE CASCADE

If this rule is applied while creating the child table we can delete a record in a master table or update the reference key value of the master table, Even if there are child records in the child table. But along with it the child table records will be deleted if it is delete & Foreign key value will be update if it is update.

ON DELETE SET NULL AND ON UPDATE SET NULL:

In this case also deleting or updating reference key values of master table is possible when corresponding child records or Existing in the child table but in case of delete or update in master table the corresponding child records foreign key value changes to NULL. Note:- If we want to impose these rules the foreign key column should not contain a NOT NUL constraint on it.

ON DELETE SET DEFAULT AND ON UPDATE SET DEFAULT:

This is same as SET NULL but in this case child tables foreign key value gets updated with default value of that column. Where as if a default value is not Provided for the column NULL is the default, default value

IMPOSING A DELETE RULE OR UPDATE RULE WHILE CREATING THE CHILD TABLE:

Delete rules and update rules were not imposed on master table, they are imposed on the child table that to on the foreign key column as following.

Syntax:- <column> <datatype> constraint <const Name> References <MT Name> (<RK.Col>)
on Delete {NoAction / cascade / set NULL / set Default}
on update {NoAction / cascade / Set NULL / set Default}

Ex:- Create table Employee (Empno Int, Ename varchar(50), Job Varchar(50), Salary Decimal (7,2), Deptno INT constraint Deptno-fk References Department (Deptno) on Delete cascade on update cascade)

Providing Default Value for a column:

→ The default, default value for any column is NULL, provided the column is not imposed with a NOT NULL constraint.

→ At the time of creating the table we have a chance of giving our own default value on the column. so that when we insert a record into the table without specifying a value to that column automatically the default value comes into picture and inserted into the column. Where as if we provide a value that value only will be inserted.

Ex:- Create Table Employee (Empno int, Ename varchar(50), Job varchar(50), Salary Decimal (7,2) Default 5000, Deptno int constraint Deptno-fk References Department (Deptno) on Delete cascade on Update cascade)

Ex- Deptno int Default 40 constraint Deptno_Fk References Department (Deptno) on delete set default on update set default)

20 June 13 Thursday

SELF REFERENTIAL INTEGRITY:

This is same as the Referential integrity we have learnt earlier. In earlier cases we are binding one column of a table with another column of another table where as in Self referential integrity we bind a column of a table with another column of same table i.e both foreign key and primary key will be present in one table only.

Primary key column	EMPLOYEE		Foreign key column
Empno	Ename	Job	Mgn
1000	Naveen	President	NULL
1001	Scott	manager	1000
1002	Smith	salesman	1001
1003	Blake	Clerk	1001
1004	Raju	manager	1000
1005	Ajay	Analyst	1004
1006	James	clerk	1010 → Invalid

In the above table we are binding the column mgn(Foreign key) with another column of the same table i.e Empno(Reference key) to verify the value entered into mgn column to be existing under Empno column

→ Creating Employee table using all constraints in column level

```
Create Table Employee
(
Empno Int constraint Empno-pk primary key, Ename
varchar(50) NOT NULL, Job varchar(50), mgn Int constraint
mgn-FK References Employee (Empno), salary Decimal(7,2)
Default 3000 constraint sal-ck check (salary Between
5000 and 15000), Deptno Int constraint Deptno-FK Reference
Department (Dept-no)
)
// Inserting Records to the above table---
```

IDENTITY PROPERTY:-

We can apply this on a table for creating a identity column for the table.

Syntax :- Identity (seed, increment)

↓ These two are optional.

Seed is the value ie used for the very first row that is loaded into table.

Increment is the incremental value i.e added to the identity values of the previous row that was loaded.

Note:- you must specify both seed and increment or neither, It neither is specify, the default is (1,1)

→ Creating table by using Identity function

```
Create Table Suplier
(
Sid int Identity (101, 1),
Sname varchar(50),
```

Goods varchar(50),
Location varchar(50),
);

→ Inserting values :-

Insert Into Suplier (Sname, Goods, Location) values ('AMIT', 'Cool Drink', 'Hyderabad')

Note:- We can't insert a value into the identity column Explicitly, but we can insert by setting a Identity_Insert Property as ON default is OFF which will not allow Explict inserting.

→ Identity property doesn't promise any uniqueness to the column values. If we want uniqueness we should explicitly impose either unique or Primary key constraint on that column.

→ Identity function can generate only Integer values can't generate alpha Numeric values.

→ When an Identity column is existing For a table with Frequent deletion gaps will occur between Identities values. So, when Frequent deletion on being done on a table do not use the identity property. However it is still possible to fill the existing gap by setting the Identity_Insert property as ON

Syntax :- Set Identity_Insert <Tname> ON

When we set the property as ON it will allow to insert values explicitly into the Identity column.

Ex:- Set Identity_Insert Supplier ON
Insert Into supplier (Sid, Sname, Goods, Location)
Values (50, 'AAA', 'Cool Drinks', 'Hyderabad')

If we want to restrict inserting values into Identity explicitly set the property as OFF again as following.

Ex:- Set Identity_Insert Supplier OFF.

ALTER TABLE COMMAND:

By using the alter Table command we can perform the following actions on a table.

1. we can Increase/Decrease the width of a column.
2. we can change the data type of a column.
3. we can change the NOT NULL to NULL or NULL to NOT NULL
4. we can add a new column
5. we can drop a existing column
6. we can add a new constraint
7. we can drop a existing constraint on a table.
8. Disable/re-enable check constraint of a table.

Example:-
```
Create Table Student
(
Sno Int,
Sname varchar(50)
)
```

→ Syntax to perform first three operations

Syntax:-

ALTER table <Tname> ALTER column <colname> <dtype>
[<width>] [NULL/NOT NULL]

// To change the width of a column Sname on student table

Ex:-
```
Alter Table Student Alter column
Sname varchar(100)
```

NOTE:- When we increase the width of a column we don't have any problem but while decreasing the width, if the table contains data in it we cannot decrease the width less than the max existing characters in the column.

// changing the data type of a column.

Ex:- Alter Table Student Alter Column
Sname nvarchar(100)

// changing the column Null to Not Null

Ex:- Alter Table Student Alter Column
Sno Int Not Null

// changing Not Null to Null

Ex:- Alter Table Student Alter Column
Sno Int Null

Friday
21/06/2013

Adding a New column under a table:

Syntax:- Alter Table <Tname> Add <col name> <dtype> [Not Null] [Constraint <const name>] <cons type>

Eg:- Alter Table Student Add class int

Alter Table Student Add Grade int Not Null

Alter Table Student Add Fees Decimal(7,2) Constraint Fees_ck check (Fees > 3000)

→ We can drop an existing column

Alter Table <Tname> Drop column <columnlist>

Ex: Alter Table Student Drop Column Grade.

→ We can add a new constraint. (optional)

Alter Table <Tname> Add [constraint <consName>] <consType>

(<conslist>)

→ Alter Table Student Add constraint Fees_Default Default 5000 for Fees.

Note:- Default is not a direct constraint but if we wanted to Provide can give a default value for any existing column of the table we can add it as a constraint on the column.

→ Alter Table student Add Constraint sname_uq unique (sname)

Adding Primary Key on a Existing Column:- If we

If we wanted to add a primary key constraint on any existing column of the table first the column must be a Not Null column then only Primary key constraint can be added.

Ex:- Alter Table Student Alter column sno int Not Null

Alter Table student Add constraint sno_PK primary key (sno);

We can drop an existing constraint on a table

Alter Table <Tname> Drop Constraint <cons Name>

Alter Table student Drop constraint sname_uq

Alter Table Student Drop constraint sno_PK

- Disable / re-enable Check constraint of a table:

Syntax:- Alter Table <Tname> Nocheck / check Constraint <ConsName>

Ex:- Insert into Student (Sno, Fees) values (101, 2500)

--Will raise an error.

Insert into stud

Alter Table Student Nocheck constraint Fees_CK

-- Disabling a constraint

Insert into Student (Sno, Fees) values (101, 2500)

--Will be inserted.

Re-enable:

Alter Table Student Check constraint . Fees_CK

--- Enabling a constraint.

Insert into student (Sno, Fees) values (102, 2500)

-- Will raise on Error again

DROPPING A TABLE: COMMAND:

Syntax:- Drop table <Tname>

Ex:- Drop table Student

Note: when a table is dropped all the dependent constraint on the table also gets dropped.

We cannot drop a master table directly...

TRUNCATE TABLE COMMAND:

Syntax:- Truncate Table <Tname>

If we Truncate a table all the rows of the table without logging them

Truncating a table is similar to delete statement with no where clause. However Truncate table is faster and uses fewer system and Transaction log resources.

We can Recover the data deleted using where but no possible in case of truncate

Where clauses can be applied in case of delete but not in case of truncate.

If a table is truncated identity column of the table is reset to Seed value. Where as delete will not do that.

If we truncate the table it removes all rows from the table, but the table structure columns, constraints remain unchanged.

We cannot truncate the data in the master table.

DML

SELECT COMMAND:-

This command is used for retesiving a data from a table (or) tables.

Syntax:- Select * |<collist> From <Tname> [<clauses>]

Clauses:

- Where
- GroupBy
- HAVING
- Order By

{ WWW.bangarraju.Net Link - Demo Tables }

22/06/2013

Dept : Deptno, Dname, Loc

Emp: Empno, Ename, Job, mgn, Hiredate, Sal, Comm, Deptno

BUILT IN FUNCTIONS:

Sql server provides number of built in function like mathematical function, String function, data type functions, Datetime function, conversion function etc for helping us in presenting as well as retriving the data

MATHEMATICAL FUNCTION:

1) ABS(n): "n" is a numeric expression. It returns a positive integer value of the given expression "n"

Eg:- select Abs(100), Abs(-100)

op 1 100 100

2) CEILING(n): Returns the smallest integer greater than or Equal to the special numeric expression.

Eg:- Select ceiling (15.3), ceiling (-15.6);

3) FLOOR(n): Returns the largest integer less than or Equal to the specified numeric expression.

Eg:- select Floor (15.3), Floor (-15.3)

4) LOG(n,[,base]): Returns the natural logarithm value of the specified expression.

Note:- By depault Log function returns the natural logarithm The natural Logarithm is to the base "e". you can also change the base of the Logarithm to another value using the optional base parameter

Eg:- select Log(10), Log(10,2), Log(10,10)

5) $Log_{10}(n)$: Returns the base-10 logarithm of the specified Expression.

Eg:- select $Log_{10}(10)$; op: 1

6) PI(): Returns the constant value of PI

Eg:- Select PI();

7) power(n,m): Returns the value of the specified value of to the specified power m

Eg:- Select power(2,4)

8) RAND([Seed]): Returns a pseudo-random float value from 0 to 1 exclusive. Seed is an integer expression. If not specified, the SQL Server Database Engine assign a seed value at Random. For a specified seed value the result returned is always the same.

Eg:- 1) Select Rand(); -- Random value changes each time
2) Select Rand(100) -- Same value generates always.
3) select Rand(), Select(100) -- Same value will generates for both everytime we execute.

9) ROUND(n, Length [, function]);- Returns a numeric value rounded to the specified length on precision.

Length is the precision to which the expression is to be rounded when length is a positive number of Expression is rounded to number of decimal positions specified by length. when length is a -negative number expression is rounded on the left side of the decimal point as specified by length. If it is positive.

Eg:- Select Round(156.789, 2), Round(156.784,1), Round(156.789, 0), Round(156.789, -1), Round(156.789, -2);

Note:- function is the type of operation to perform when function is not specified or has a value of zero(0) [default] expression is rounded. when a value other than 0 is specifieds expression is functional.

Eg:- Select Round (156.789, 2, 1)

Round (156.789, 1, 1)

Round (156.789, 0, 1)

Round (156.789, -1, 1)

Round (156.789, -2, 1)

// Generates a Random no between 1 to 100.

Eg:- Select Round (Rand() * 1000, 0)

10) SIGN (n): Returns the sign of the specified expression which will be (+1) <positive> (0) <0> or (-1) <negative>

Eg:- Select sign(10), sign(0), sign(-10),

o/p - 1 0 -1

11) SQRT(n): Returns the square root of the specified value.

Eg:- Select SQRT(4);

12) SQUARE(n): Returns the Square of the specified value

Note:- Apart From the above we are also provided with Trignometric Function like ACOS, ASIN, ATAN, ATAN2, COS, COT, SIN, TAN for which we need to pass n as a Parameter where as n represents the angle.

STRING FUNCTION:

1) ASCII (s): 's' store string Expression. Returns the ASCII Code value of the left most character in the expression.

Eg:- Select ASCII ('A'), ASCII ('Hello')

o/p = 65 72

2) CHAR(n): Converts given ASCII code to a character

Eg:- Select char(90) o/p - Z

3) NCHAR(n): Returns the unicode character for the specified integer

Eg:- Select Nchar(256); o/p: Ā

--n can be greater than 255 also.

4) LEN(s): Returns the number of characters in the specified string option excluding trialing spaces

Eg:- Select Len('Hello'), Len('Hello '), Len('Hello')

Eg:- Select * From Emp where Len(Ename) = 5

5) CHARINDEX (expToFind, expTo search [, start_Location]):

Searches for an expression in another expression and return its starting index position is Fand.

Eg:- Select charindex ('w', 'Hello world') → 8

charindex ('o', 'Hello world') 15

Charindex ('o', 'Hello world', 6) 18

charindex ('a', 'Hello world') 0

→ Write a Query to get the List of Employee whose name contain char 'a' in it.

Select * From Emp where

Charindex ('a', Ename) > = 1

6) CONCAT (string_value1, string_value2 [, string_valueN]):

Returns a string i.e the result of concatinating 2 or more string values.

Eg:- Select concate ('HELLO', 'HOW', 'ARE', 'YOU');

7) FORMAT(value, Format [, culture]): Returns a value formatted with the specified format [d, D, N, G, C] and in optional culture in SQL server 2012. use the FORMAL function for Local-aware formatting of date/time, number and currency values.

Date Formatting:-

```
Select GetDate(), Format (GetDate(), 'D')
Select GetDate(), Format (GetDate(), 'D', 'hi-IN')
Select GetDate(), Format (GetDate(), 'D', 'hi-IN')
```

Number formatting:-

```
Select Format (123456T., 'N')
Select Format (123456T, 'N', 'hi-IN')
```

Currency Formatting:-

```
Select Format (123456T, 'C')
Select Format (123456T, 'C', 'hi-IN')
```

Monday
24/06/2013

Ex:- Select Empno, Ename, Job
Format (Hiredate, 'd', 'hi-IN') AS DOJ,
Format (Sal, 'c', 'hi-IN') AS Salary From Emp.

LEFT(S,n): Returns the left part of the string with the specified number of characters

Ex:- Select Left ('Hello', 3)

Hel

→ Write a query to retrive the list of Employees whose name start with the character S

Select * From Emp Where Left(Ename, 1) = 'S'

RIGHT(S,n): Returns the right part of the string with the specified number of characters.

Ex:- Select Right ('Hello', 3)

llo

→ Write a query to get the list of Employees whose name ends with the character y

Select * from Emp Where Right (Ename, 1) = 'y'

→ Write a query to get the list of Employees whose name contain character 'a' in 3 place.

Select * From Emp Where Right (Left (Ename, 3), 1) = 'a'

SUBSTRING (S, Start, Length): Returns Part of a character binary Text or image Expression from the given Expression.

Select Substring ('Hello', 2, 3)

→ Write a query to get the list of Employees whose name contains character 'a' in 3 place

Select * from Emp where substring ('Ename', 3, 1) = 'a'

→ Write a query to get the list of Employees whose name Contains character 'a' in it.

Select * From Emp Where CHARINDEX ('a', Ename) 1 = 0

Select * From Emp Where CHARINDEX ('a', Ename) >= 1

LOWER(s): Returns an Expression after converting data to Lowercase.

Select Lower ('Hello')

Ans hello

Select Empno, Lower (Ename) as Ename, Job, Sal From Emp

UPPER(s): Returns an Expression after converting data into uppercase.

Select Upper ('Hello')

Ans HELLO

Select Empno, Upper (Ename) as Ename, Job, Sal From Emp

LTRIM(s): Returns a character Expression after it removes the Leading blanks

Select Len (' Hello')

8

Select Len (LTrim (' Hello'))

5

Select Len (' Hello'), Len (LTrim (' Hello'))

5 5

RTRIM(s): Returns a character string after truncating all trialing blanks.

Select Len('Hello '), Len Rtrim('Hello '))

o/p 5 5

REPLACE(s, string_search, string_replacement):- Replaces all occurances of the search string value with replacement string value.

Select Replace('Hello world', 'l', 'x') o/p:- Hexxo worxd

Select 'John Smith
John smith

Select * from Emp where Ename = 'John Smith'
Select * from Emp where Ename = 'Johnsmith'

Eg:- Select * from Emp where Replace('Ename, ' ', '') = 'Johnsmith'

o/p:- Johnsmith.

REPLICATE(s, n): Repeats a string value for specified number of times.

Select Replicate('Hello', 3)

o/p HelloHelloHello

REVERSE(s): Returns a reverse order of a string value.

Select Reverse('Hello')

olleH

SOUNDEX: Returns a 4 character (soem SOUNDEX) code to Evaluate the similarity of two strings

Eg:- Select Soundex('color'), Soundex('colour')
Select soundex('Smith'), Soundex('Smyth')
Select- Soundex('Greed'), soundex('Greene')

Ans:- C460
S530
G650

C460
C630
C650.

Ename = Smith, Smyth

Eg:- Select * from Emp where Ename = 'Smith'

Select * from Emp where Ename = 'Smyth'

Select * from Emp where Soundex(Ename) = Soundex('Smith')

Select * from Emp where (OR) Soundex(Ename) = Soundex('Smyth')

NOTE:- Soundex converts an alphanumeric string to a four character code that is based on how the string sounds when spoken the first character of the code is the first character of the given expression converted to uppercase. The second to fourth characters of the code are numbers that represent the letters in the expression.

(ex-0)

DIFFERENCE(s_1, s_2): Returns an integer value that indicates the difference between the Soundex value of two character Expression.

NOTE:- Soundex code from different strings can be compared to see how similar the strings sound when spoken. The difference function performs a soundex on two strings and returns an integer that represents how similar the Soundex codes are for those things.

Eg:- Select Difference ('color', 'colour')

Ans 4 — quite similar.

Select Difference ('India', 'America')

Ans 1 — quite opposite.

The Result here can be from 1 to 4 where,
4 indicates the 2 strings are similar (strong similarity)
1 indicates the 2 strings are (weak similarity)

SPACE(n): Returns a string of Repeated Spaces.

Select 'Hello' + Space(5) + 'World'

'Hello World'

STUFF(S, start, length, replace_string): It inserts a string into another string by deleting the specified length of characters in the first string at the start position upto the specified length and inserts the second string into the first string

Ex:- Select Replace('Hello World', 'l', 'x')

Hexxo Woxld

Select Stuff('Helloworld', 3, 2, 'x')

Hexo world

Select Stuff('Hello . World', 6, 5, '')

Hello world.

DATA TYPE FUNCTIONS:

DATALENGTH(Expression): Returns the number of bytes used to represent any Expression

Ex: Select * From

Select DataLength(Empno), DataLength(Ename) From Emp

Empno Ename (varchar)

Ex:- 4 5
6

fixed length. variable length.

Ex:- Select DATALENGTH (100), Data Length ('Hello')

4 5

IDENT_CURRENT ('table_name'): Returns the last identity value generated for a specific table

Ex:- Select IDENT_CURRENT ('Supplier')

Op - 111

IDENT_SEED ('table_name'). Returns the original seed value that was specified when an identity column in a table is created.

Ex:- Select IDENT_SEED ('Supplier')

Ans: 101

IDENT_Incr ('table_name'): Returns the increment value specified during the creation of an identity column in a table.

Ex:- Select Ident_Incr ('Supplier')

Ans:- 1

25/06/2013

DATE AND TIME FUNCTIONS:

GETDATE(): Returns a datetime value that contains the date and time of a computer on which the instance of SQL Server is running.

GETUTCDATE():- Returns a datetime value that contains the date and time of the computer on which the instance of SQL server is running. The date and time is returned as UTCtime (Coordinated Universal Time). This is same as GMT.

SYSDATETIME: Returns a date time 2(7) value that contains the date and time of the computer on which the instance of SQL Server is running.

SYSDATETIMEOFFSET: Returns a date time offset (7) value that contains the date and time of the computer on which the instance of Sql server is running. The time Zone offset is included.

SYSUTCDATETIME: Returns a datetime2(7) value that contains the date and time of the computer on which the instance of SQL server is running.

Eg:- Seled GetDate(), GetUTCDate(), SYSDATETIME(), SYSDATETIMEOFFSET(), SYSUTCDATETIME().

DATENAME(datepart ~~date~~ date): Returns a character string that represents the specified datepart of a specified date.

Datepart is the value we want from the given date which can be any of the following.

datepart	Abbreviations
year	yy, yyyy
quarter	qq, q
month	mm, m
day of year	dy, y
day	dd, d
week	wk, w
weekday	dw, w
hour	hh
minute	mi
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns
Tzoffset	tz

Eg:- Select Datetime (mm, GetDate()) June

Select Datetime (tz, SYSDATETIMEOFFSET()) +07:00

Select Datetime (yy, GetDate()) 2013.

→ Write a Query to get the list of Employees joined in the month of April.

Select * From Emp where DateName (mm, Hiredate) = 'April'

DATEPART (datepart, date): Returns an integer that represents the specified datepart of the specified date.

Eg: select * from Emp where Datepart (mm, Hiredate) = 04

Note: DateName and Datepart functions are very similar the only difference is the first one gives the result as varchar and second one gives the Result as int.

DAY (date): Returns an integer representing the day of the month of the specified year date

Eg:- Select Day(GetDate())

MONTH (date): Returns an integer that represents the month part of a specified date.

Eg: Select Month (GetDate())

Eg:- Select * from Emp where Month(Hiredate) = 04.

Year (date): Returns an integer that represents the year part of the specified date.

Eg:- Select year (GetDate()) → 2013

Select year ('10/24/1978')

Select year ('1978/10/24') } 1978

66

Note: When we Enter the date manually it must be either in mm/dd/yyyy format (or) yyyy/mm/dd

EOMONTH(start_date, [, month_to_add]): Returns the last day of the month in the specified date

Eg:- Select EOMONTH(GetDate()) 2013-06-30
Select EOMONTH(GetDate(), 15) 2014-09-31
Select EOMONTH(Ge Date(), -18) 2011-12-31

DATEDIFF(datepart, startdate, enddate) Returns the count of the specified datepart boundaries across the specified startdate and enddate

Select DateDiff(yy, '12/25/1900', GetDate()) 113
Select DateDiff(mm, '12/25/1900', GetDate()) 1350

→ Write a query to find the number of years each employee is working in the organisation

alternate [Select * from Emp where DateDiff(yy, HireDate, GetDate())]

→ Select Empno, Ename, Job, Sal, DateDiff(yy, HireDate, GetDate()) as Seniority, from Emp

→ Write a query to get the list of Employees worked for 32 years.

Select * From Emp where DateDiff(yy, HireDate, GetDate()) = 32

DATEADD(datepart, number, date): Returns a new datetime value by adding an interval to the specified datepart of the specified date.

Eg:- Select DATEADD(dd, 168, GetDate()) 2013-12-10 13:01:39:727

Eg:- Select DATEADD(mm, 78, GetDate())

2019-12-25 13:03:32.340

DATEFROMPARTS (year, month, day): Returns a date value for the specified year, month and day.

DATETIMEFROMPARTS (year, month, day, hour, minute, seconds milliseconds)
Returns a datetime value for the specified date and time.

TIMEFROMPARTS (hour, minute, seconds, fractions, precision).
Returns a time value for the specified time and with the specified Precision.

Eg:- Select DATEFROMPARTS (1918,07,17)
Select DATETIMEFROMPARTS (1918, 07, 17, 12, 50, 30, 00)
Seled TIMEFROMPARTS (11,53,36,00,3)
Seled TIMEFROMPARTS (11,53,36,00,7)

Eg:-

ISDATE (Expression): Returns 1 if the Expression is a valid date; otherwise Returns 0.

Eg:- Select ISDATE ('06/25/2013'), ISDATE('25/06/2013'), ISDATE('2013/06/25')

→ O/p 1 0 1

SETDATEFORMAT [format]: Sets the order of the month, day, and year date parts for interpreting date in a session.

- Format can be mdy (default), dmy, ymd, ydm, myd, and dym.

Eg:- Select SETDATEFORMAT dmy
Select ISDate ('06/25/2013'), ISDATE('25/06/2013'), ISDATE ('2013/06/25') → O/p 0 1 0

Wed 26/06/2013

CONVERSION FUNCTIONS:

This functions are used for converting a value from one datatype to other.

CAST (expression AS data_type [(length)]): Converts an expression of one data type to another.

Eg:- Select cast(GetDate() as varchar(12)) o/p

Length is optional. The default value of length is 30 {Jun 26 2013}

eg:- Select cast (123.456 as int) o/p 123

CONVERT(data_type [(length)], expression [, style]):

Convert is same as cast o/p Jun 26 2013

Eg:- Select Convert (varchar(12), GetDate())

Select convert (varchar int, 123.456) 123.

Length is an optional integer that specifies the length of target data type. The default value is 30.

Style is an integer expression that specifies how the convert function has to translate the expression.

If style is NULL NULL is returned.

When expression is a date or a time data type, style can be one of these values:

100 to 114, 120, 121, 126, 127, 130, 131 (With century)

(or)

1 to 8, 10, 11, 12, 14 (without century).

Note: style 0 or 100, 9 or 109, 13 or 113, 20 or 120, and 21 or 121 always return the century (yyyy)

Ex:- Select convert (varchar(12), GetDate(), 101) 26/06/2013
Select convert (varchar(12), GetDate(), 102)
Select convert (varchar(12), GetDate(), 103)
Select convert (varchar(12), GetDate(), 3) 26/06/13

When expression is money (or) Small money style can be one of these values 0, 1, 2, 126 (Equivalent to style 2 use it when converting to nchar (or) nvarchar)

Eg:- Select convert (varchar, sal, 0) From Emp
Select convert (varchar, sal, 1) From Emp
Select convert (varchar, sal, 2) From Emp

O/P
5,000
5000

LOGICAL FUNCTIONS:

1. CHOOSE (INDEX, val_1, val_2 [, val_n]): Returns the item at the specified index from a list of values.

O/P - 30.

Eg:- Select choose (3, 10, 20, 30, 40, 50)

2. IIF (boolean_expression, true_value, false_value);- Returns one of two values, depending on whether the Boolean expression evaluates to true or false.

O/P
Hai
Hello

Eg:- Select IIF (10<20, 'Hello', 'Hai')
Select IIF (10>20, 'Hello', 'Hai')

Thursday
27/06/2013

SYSTEM FUNCTIONS:

HOST_ID(): Returns the workstation identification number. The workstation identification number is the process ID (PID) of the application on the client computer that is connecting to SQL server.

Eg:- Select Host_ID()

3944

HOST_NAME(): Returns the workstation name.

Eg:- Select Host_Name()

Server

ISNUMERIC (expression): Determines whether an expression is a valid numeric type.

NEWID(): creates a unique value of type uniqueidentifier

Ex:- Select ISNumeric(10.56), ISNumeric('Hello'), ISNumeric('1000')

o/p - 1 0 1

Note:- We use the NewID function for inserting a values into a column with a unique identifier datatype.

ISNULL (check_expression, replacement_value): Replaces NULL with the specified replacement value.

Eg:- Select ISNULL (100, 200), ISNULL (NULL, 200)

o/p 100 200

Eg: Write a query to get the total sal of all the Employee

Select Empno, Ename, Job, Sal, comm, Sal + ISNULL(comm, 0) as Total sal From Emp

Note: Any arithmetic operation performed on a null value will return a null value again.

COALESCE (expression [,...n]): Returns the first non null Expression among its arguments. This is same as ISNULL but can be given with multiple expressions.

Eg:- Select coalesce (Null, Null, 10, 20, Null, 30)

10.

Select Empno, Ename, Job, Sal, Comm, Sal + Coalesce (Comm, 0) as Total sal From Emp

NULLIF (Expression, expression): Returns a null value if the two specified expressions are equal or else returns first expression

Eg:- Select NULLIF (10, 20), NULL (10, 10)

o/p to NULL

CASE: Evaluates a list of conditions and returns one of multiple possible result expressions. The case expression has two formats

1. The simple case expression which compares an expression with set of values to determine the result.

Syntax: CASE input expression
WHEN value THEN result_expression [...n]
[ELSE else_result_expression]
END.

Q:- Write a query to give a comment column on the empdept table as President Bigboss
Manager Boss
Analayst Scientist
others Employee

Sol:- Select Empno, ename, Job
case Job
When 'president' then 'Bigboss'
When 'manager' then 'Boss'
When 'Analyst' then 'Scientist'
Else 'Employee'
End As comment
From Emp

Q:- Write a query to give a comment column on the employee table (Self)

x [Select Empno, ename, Sal,
Case Sal
When Sal > 3000 Then 'Above Target'
When Sal = 3000 Then 'On Target'
Else 'Below Target'
End AS Comment] x

Select Empno, ename, Sal,
Case sign(Sal - 3000)
When 1 Then 'Above Target'
When 0 Then 'On Target'
Else 'Below Target'
End AS comment
from Emp.

Q. The Searched CASE Expression which evaluates a set of boolean expression to determine the result.

Syntax:- CASE
WHEN <condition> THEN result_expression [...n]
[ELSE else_result_expression]
END.

Ex:- Select Empno, Ename, Sal,
CASE
WHEN Sal > 3000 THEN 'Above Target'
WHEN Sal = 3000 THEN 'ON TARGET'
Else 'Below Target'
End AS comment
From Emp.

META DATA FUNCTIONS:

APP_NAME(): Returns the application name for the current session.

Eg:- Select APP_NAME()

O/p - Microsoft SQL Server - Query (Visual Studio)

COL_LENGTH('table', 'column'): Returns the defined length of a column in bytes

Eg:- Select COL_LENGTH('Emp', 'Ename') 100

Select COL_LENGTH('Emp', 'Empno') 4

DB_ID([database_name]): Returns the database identification (ID) Number

Eg- Select DB_ID(), DB_ID('SQL'), DB_ID('Master')

O/p- 7 7 1

DB_NAME([database_id]): Returns the database name for the given database id.

Ex:- Select DB_NAME(1), DB_NAME(2)

O/p- Master tempdb.

OBJECT_ID('object_name'): Returns the database object identification number of a object.

Eg:- Select OBJECT_ID('Emp'), OBJECT_ID('DEPT')

O/p - 46-9576711 437576597

OBJECT_NAME([object_id]): Returns the object name for the given object id.

Eg:- Select OBJECT_NAME(469576711)

O/p- Emp

COL_NAME (table_id, column_id) :- Returns the name of the specified column from the corresponding table identification Number and column index number.

Eg:- Select col_name (OBJECT_ID ('Emp'), 1)
o/p Empno.

RANKING FUNCTIONS:

Top clause:- using the "Top clause" we can get the top records of a table.

Eg:- Select Top 5 * FROM Emp
Select Top 5 * FROM Emp orderBy Empno Desc
(Sal) Select Top 5 * FROM Emp orderBy sal Desc

RANK() OVER ([partition_by_clause] <order_by_clause>)

Returns the rank of each row in the table.

Eg:- Select Empno, Ename, Sal, Rank() over (order by sal) from Emp default - asc
Select Empno, Ename, sal, Rank() over (order by sal) From Emp.

→ Write a query to get the top 7 salaried Employees of the table with their ranks.

Sol:- Select Top 7 Empno, Ename, Sal, Rank() over (order by sal Desc) from Emp.

Partioned by clause can be applied for dividing or grouping the data basing on any column.

Eg:- Select Empno, Ename, Sal, Deptno, Rank() over (Partition by Deptno order by sal Desc) from Emp
o/p- Dept-wise

Select Empno, Ename, sal, Job, Rank() over (partition by Job order by sal Desc) from Emp. (Jobwise) o/p

DENSE_RANK() OVER ([partition_by_clause] < order_by_clause >): Returns the rank of rows in the table, without any gaps in the ranking

Eg: Select Empno, Ename, sal, Dense_Rank() over (order by sal Desc) from Emp

Select Empno, Ename, Sal, Deptno, Dense_Rank() over (order by Partition by deptno order by sal Desc) from Emp

Select Empno, Ename, sal, Job, Dense_Rank() Over (partition by Job order by sal Desc) from emp. (it will take the duplication)

ROW_NUMBER() OVER ([partition_by_clause] < order_by_clause >): Returns the Sequential number of a row in the table being retrieved starting at 1 for the first row of Each partition

Eg:- Select Empno, Ename, sal, ROW_NUMBER() OVER (order by sal desc) from Emp

Select Empno, Ename, Sal, Deptno, ROW_NUMBER() over (partition By deptno order By sal Desc) from Emp

NTILE (Integer Expression) OVER ([partition_by_clause] < order_by_clause >): Distributes the rows into a specified number of groups. The groups are numbered, starting at one. For each row, NTILE returns the number of the group to which the row belong.

Eg:- Select Empno, Ename, Job, Sal, Deptno, NTILE(3) order (order By Sal Desc) From Emp.

28/06/2013 Friday

AGGREGATE FUNCTIONS: Aggregate function perform a calculation on a set of values and returns a single value. Except COUNT, aggregate functions ignore null values.

DISTINCT Keyword: If we use this keyword on a column within a query it will retrieve the values of the column without duplicates.

Eg:- Select Distinct Job From Emp
Select Distinct Deptno From Emp.

COUNT([[DISTINCT] Expression] | *) :- Returns the no. of items in a group.

Select count(*) From Emp

Note:- When we use * in count it returns no. of rows in the table. In place of *, if we use a column name it will return the no of NOT NULL values in that column.

o/p
15
Select count(Ename) From Emp 4
Select count (comm) From Emp 3
Select Count (Distinct Deptno) From Emp 5
Select Count (Distinct Job) From Emp

If Distinct is used on the column will ignore duplicates

COUNT_BIG([[Distinct] expression] | *) :- Returns the number of items in a group

Note:- COUNT_BIG works like the COUNT function only. The only difference between the two functions is the RV return value. COUNT_BIG returns a ~~beig~~ bigint datatype value where as COUNT returns an int datatype value.

Sum ([DISTINCT] Expression) :- Returns the sum of all the values (or) only the distinct values in the expression. Sum can be used with numeric-columns only. Null values are ignored.

Eg:- Select Sum(sal) From Emp
Select Sum(comm) From Emp

Select Sum(Deptno) From emp
Select Sum(Distinct Deptno) From Emp.

AVG ([DISTINCT] expression) :- Returns the average of the values in a group. Null values are ignored.

Eg:- Select sum(sal), Avg(sal) From Emp
Select sum(comm), Avg(comm) From Emp.

MIN ([DISTINCT] Expression): Returns the minimum value in the expression.

Eg:- Select MIN(Sal) From Emp
Select MIN(Hiredate) From Emp.

MAX ([DISTINCT] expression): Returns the maximum value in the expression

Eg:- Select MAX(Sal) From Emp
Select MAX(Hiredate) From Emp

STDEV ([DISTINCT] expression) :- Returns the statistical standard deviation of all values in the specified column Expression.

VAR ([DISTINCT] Expression) :- Returns the statistical variance of all values in the Specified Expression.

OPERATORS: An operator is a symbol specifying an action that is performed on one or more expressions. The following lists the operator categories that Sql Server supports.

1. ARITHEMATIC OPERATORS: Arithematic operators perform mathematical operations on two expressions of same or different datatypes of numeric data

+, -, *, /, %

→ Write a Query to get the total investment made for salaries every month.

```
Select Sum(sal)+sum(comm) AS total salaries From Emp

Select Empno, Ename, sal, comm, sal + ISNULL(comm,0) as
Total Salary From Emp

Select Empno, Ename, sal, comm, (sal + ISNULL(comm,0)) * 12
as AnnualSalary From Emp
```

ASSIGNMENT OPERATOR: The Equal sign (=) is the only assignment operator.

COMPARISION OPERATOR: Comparision operators test whether two Expression are the same, we can use Comparision operators on all expressions except expressions of type text, ntext or image.

=	Equals
>	Greaterthan
<	Less than
>=	Greater than or Equal to

<=	Less than or Equal to	
<>	Not Equal to	
!=	Not Equal to (NOT ISO Standard)	(This we will use in SQL only)
!<	Not less than (NOT ISO Standard)	
!>	Not greater than (NOT ISO Standard)	

→ Write a Query to get the list of Employees earning a salary greater than 3000.

Sol:- Select * From Emp where Sal > 3000

Select * From emp Where Sal !< 3000 (Here 3000 will be taken into Consideration)

!<	Equals	>=
!>	Equals	<=

COMPOUND OPERATORS:- Compound operators execute some operation and set an original value to the result of the operation.

+=	Add Equals
-=	Subtract Equals
*=	Multiply Equals
/=	Divide Equals
%=	Modulo Equals

Eg:- update emp set Sal += 100 Where Empno = 1007.

LOGICAL OPERATORS:- Logical operators test for the truth of some condition. Logical operators like comparision operators returns a Boolean datatype with a value of TRUE, FALSE or UNKNOWN.

And: Returns true if both boolean Expressions are true.

OR: Returns true if either of the boolean Expression is TRUE

NOT: Reverses the value of any other boolean operator.

BETWEEN: Returns true if the operand is within a range

In: Returns true if the operand is equal to one of a list of Expressions.

Like: Returns true if the operand matches a pattern.

Exists: Returns true if a sub-query contains any row

SOME: Returns true if some of a set of Expressions are true.

Any: Returns true if any one of a set of expressions are true

All: Returns true if all of a set of comparision are true.

→ Write a query to get the list of employees whose Job is manager with a salary of 4000.

Sol:- Select * From Emp where Job = 'Manager' and Sal = 4000

→ Write a query to get the list of Employees who are earning a salary more than 3000 with Salesman as a Job

Sol:- Select * From Emp where Sal > 3000 and Job = 'Salesman'

→ Write a query the list of Employees whose Job is Clerk, manager or Analyst.

Sol:- Select * From Emp where Job = 'Clerk' or ~~Job =~~ Manager or Job = 'Manager' or Job = 'Analyst'.

or

Select * from Emp where Job IN ('clerk', 'manager', 'Analyst')

Select * from Emp Where Job Not IN ('Salesman', 'president')

(or)

Select * From Emp Where Job <> 'Salesman' And Job <> 'president'

→ Write a Query to get the list of Employees whose Salaries From 3000 to 5000.

Select * From ~~Where~~ Emp Where ~~Sal~~ Between 3000 and 5000

(or)

Select * From Emp Where Sal >= 3000 and Sal <= 5000

(or)

Note:- 'Between operator takes Max and min ~~oper~~ values also into consideration

Select * From Emp Where Not sal < 3000 and not sal > 5000

→ Write a Query to get the list of Employees earning a Salary that is lessthan 3000.

Select * From Emp Where Sal < 3000

(or)

Select * From Emp Where NOT Sal >= 3000

→ Write a Query to get the list of Employees whose name starts with the character 'S'.

Sol:- *(Select * From Emp Where Left (Ename, 1) = 'S')*

(or) here space will not get

Select * From Emp where Ename like 'S%']*

Select * From Emp Where Left (LTrim (Ename), 1) = 'S'

(or)

Select * From Emp Where ~~L~~trim (Ename) like 'S%'

→ Write a query to get a list of Employees whose name is Smith or Smyth.

Sol:- Select * From Emp where soundex(Ename) = soundex('Smith')

(or)

Saturday 29/06/13 Select * From Emp where Ename Like 'Sm-th'

Like operator determines whether a specific character string matches a specified pattern. A pattern can include regular characters and wildcard characters. During Pattern matching regular characters must exactly match the characters specified in the character string. However Wildcard characters can be matched with arbitrary Fragments of the character string. using wild-card characters make the Like operator more Flexible than using the Equal(=) and not Equal(!=) string comparision operators.

% : Any String of zero or More characters

_ : Any single character.

[] : Any single character specified range ([a-f]) or set ([abcdef]).

[^] : Any single character not within the specified range ([^a-f]) or set([^abcdef]).

→ Write a query to get the list of Employees whose name starts with any character between A-G

Sol:- Select * From Emp where Ename Like '[A-G]%'

(or)

Select * From Emp where Ename Like '[ABCDEFG]%'

→ Write a query to get the list of Employee whose name doesn't start with characters between a-g

Sol: Select * From Emp Where Ename Not Like '[a-g]%'

(or)

Select * From Emp Where Ename Like '[^a-g]%'

(or)

Select * From Emp where Ename Like '[^abcdefg]%'

(or)

Select * From Emp Where Ename Like '[h-z]%'

SET OPERATORS: SQL SERVER provides the following set operators which combines results from two or more queries into a single result set. Those are:

- Union
- UnionAll
- Intersect
- Except

To combine the results of two queries we need to follow the below basic rules.

1. The number and the order of the columns must be same in the all queries.
2. The datatypes must be compatible.

→ Write a Query to find out the jobs associated with deptno 20 & 30.

Sol:- Select Job From

U.

UNION: Combines the results of two or more queries into a single result set, that includes all the rows that belong to all queries in the union eliminating duplicates.

UNION ALL: Same as union but duplicates will not be eliminated.

Sol:- Select Job From Emp where Deptno = 20
UNION
Select Job From Emp where Deptno = 30

(OR)

Select Job From Emp where Deptno = 20
union all
Select Job From Emp where Deptno = 30.

INTERSECT: Returns any Distinct values that are returned by both the queries on the left and right sides of intersect operand.

→ Write a query to get the common Jobs in deptno 20 & 30.

Sol:- Select Job From Emp where Deptno = 20
Intersect
Select Job From Emp where Deptno = 30.

Except: Returns any distinct values From the left query that are not found on the right query.

→ Write a query to get the jobs that are specific to Deptno 20 when compared with Deptno 30.

Sol:- Select Job From Emp Where Deptno = 20
Except
Select Job From Emp Where Deptno = 30

String Concatenation operators: +, +=
→ (Write a query to connect a line Hello your Salary is (in place of Empno & Ename))

Eg:- Select Empno, 'Hello' + Ename + 'your salary is' +
Convert (varchar(12), Sal) as Comment From Emp

Select Empno, 'Hello' + Ename + 'your salary is' +
format (sal, 'c', 'hi-IN') as Comment from Emp

Giving alias Names to columns: We can give alias Name to any column in the select list in two different ways.

Eg:- Select Empno, Ename, sal as Salary From Emp
Select Empno, Ename, Salary = sal From Emp
// (2012)

CLAUSES: We can add these two to a Query for adding additional options like filters, grouping, arranging etc.

- Where
- GroupBy
- HAVING
- ORDER By

Where:- Specifies a search condition for the rows returned by the query so that it will restrict all the unwanted data.

Group By: used for grouping a set of rows into a set of summary rows basing on the values of one (or) more columns

→ Write a query to find out the number of employees working in the organization.

Select Count(*) From Emp

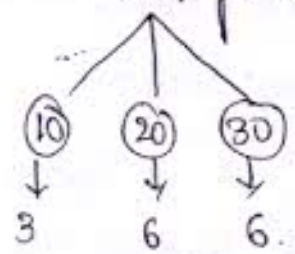
→ Write a query to find the total number of Employees belonging to each department.

Select deptno, Count(*) From Emp Groupby Deptno.

(or)

Select deptno Empcount = Count(*) From Emp Groupby deptno.

(Emp Num take place by Groupby clause)



dept	Emp C
10	3
20	6
30	6

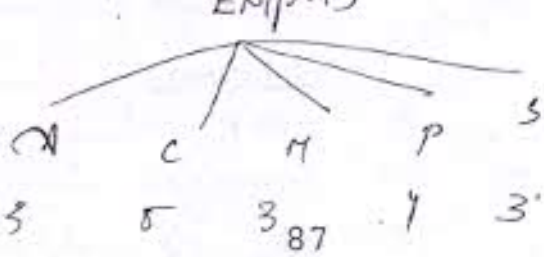
If we use Groupby clause in a query first the data in the table will be divided into different groups basing on the column specified in the Groupby clause and then executes the Group function on each Group to get the results.

→ Write a query to get no. of Employees per each job.

Sol:- Select Job, Count(*) From Emp Groupby Job

(or)

Select Job, EmpCount = Count(*) From Emp Groupby Job.



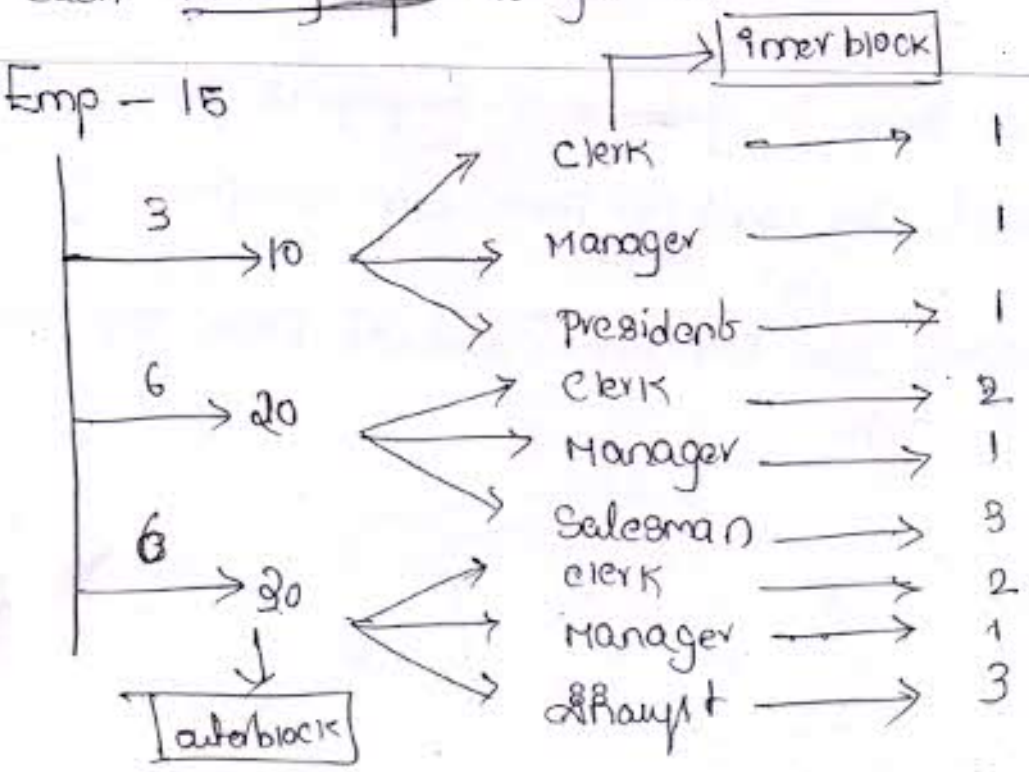
While using Group By clause in a Query, we need to follow (or) consider the following.

1. When a group function is applied on a group it returns only a singe value but each group can return a seperate value.
2. Use group-by clause only on a column which contains duplicate values, never apply it on unique columns.

→ Write a Query to get the number of employees working in each job per each department.

Sol:- Select deptno, Job, Empcount = count(*) From Emp
Group By Deptno, Job order by Deptno.

When we use multiple columns in a group by clause first, data in the table is divided basing on the first column of the groupBy clause and then each group is sub-divided basing on the second column of the groupBy clause. and then the group function is applied on each inner group to get the result



01/07/2013 | Monday

Rollup and Cube Operators:

Rollup: Generates a simple group by aggregate rows plus sub-totals and also a grand total row.

Eg:- Select count (*) From Emp

Select Deptno; count (*) From Emp Group By deptno

Select Deptno, Job count(*) From Emp Group by deptno, Job order By deptno, Job.

(or)

Select Deptno, Job count(*) From Emp Group By Rollup (Deptno, Job)

O/p:-

Deptno	Job	Count
10	Clerk	1
10	Manager	1
10	President	1
10	Null	3
20	Clerk	2
20	Manager	1
20	Salesman	3
20	Null	6
30	Clerk	2
30	Manager	1
30	Analyst	3
30	Null	6
Null	Null	15

Cube operator:- Generates a simple group by aggregate rows, rollup aggregate and cross-tabulation rows.

Eg:- Select count (*) From Emp
Select Deptno, count (*) From Emp Group by Deptno
Select Job, count (*) From Emp Group by Job
Select Deptno, Job, count (*) From Emp Group By Deptno, Job order By Deptno, Job.

(or)

Eg:- Select Deptno, Job count (*) From Emp Group By cube (Deptno, Job)

Deptno	Analayst	Clerk	Man	P	S	T
10	0	1	1	1	0	3
20	0	2	1	0	3	6
30	3	2	1	0	0	6
	3	5	3	1	3	15

O/p - Analyst 3
Clerk 5
Manager 3
President 1
Salesman 3

→ Write a Query to get the highest salary in the organization

Sol:- Select Max (Sal) From Emp

→ Write a Query to get the highest salary of Each dept

Sol:- Select Deptno, Max(Sal) From Emp Group By Deptno

→ Write a query to find out the highest sal of Each Dept, along with the name of the Empayee.

Sol: Select Deptno, Max(Sal), Ename From Emp Group By Deptno. // Invalid

Note:- The above Query will not be executed because while using groupBy clause in a statement the Select list can contain only three things in it

1. Columns that are associated with GroupBy clause
2. Aggregate (or) Group functions
3. Constants

In our above Query Ename (column) doesn't fall under any of the three. So, it cannot be used in the select list

Eg:- Select Deptno, Max(Sal), Ename From Emp Group By deptno // Invalid

Select Deptno, Max(Sal), Get-Date() From Emp Group By deptno // valid

Select Deptno, Job, Max(Sal), Hello From Emp Group By deptno, Job // valid

→ Write a Query to get the total investment being made on Salaries For all the Employees for Each Job and for Each Job in Each department.

Sol:- Select Deptno, Job, Sum(Sal) From Emp Group By Rollup (Deptno, Job).

→ Write a Query to get all the three above as well as Jobwise investment also.

Sol:- Select Deptno, Job, Sum(Sal) From Emp Group By cube (Deptno, Job).

→ Write a Query to find out the total number of clerks in the organization.

Sol:- Select Count(*) From Emp Where Job = 'clerk'

→ Write a query to find out the number of clerks in Each department

Sol:- Select Deptno, Count(*) From Emp Where Job = 'clerk' Group By Deptno.

→ Write a query to gets the number of Employees in Each department only if count is greater than 5.

Sol:- Select Deptno, Count(*) From Emp Where Count(*) > 5 Group By deptno // Invalid

Select Deptno, count(*) From Emp Group By deptno Having Count(*) > 5.

HAVING CLAUSE: This clause is also used for restricting the data. Just like Where clause.

The difference between Where clause and Having clause is.

* Where clause is used for restricting the rows before Grouping and Having clause is used for restricting the rows after grouping.
* If the restriction is associated with a aggregate Function we cannot use Where clause there. We need to use Having clause only.
* Where clause can be used for restricting individual rows Where as Having clause is used along with Group By clause to Filter (or) restrict a groups.
* Having clause cannot be applied without a group by clause.

→ Write a Query to find out the number of Employees in Each job only if the count is less than 5.

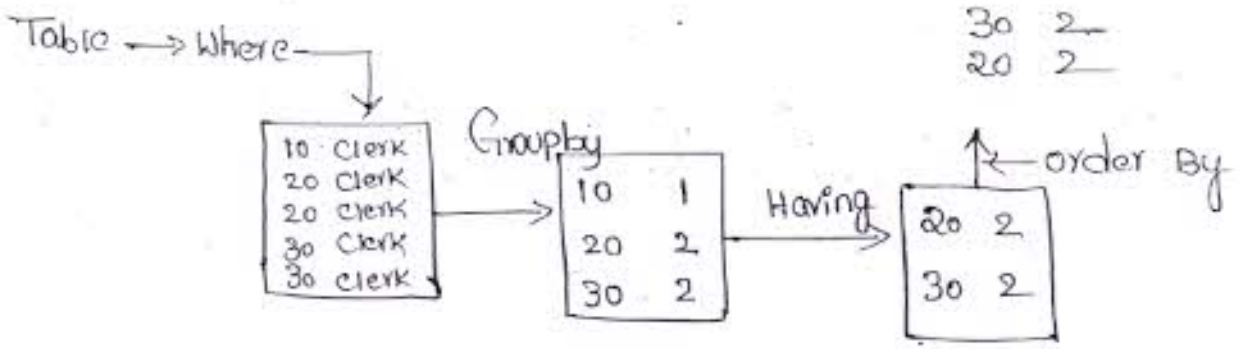
Sol:- Select Job, Count(*) From Emp ~~where Count(*)<5~~
Group By ~~Job~~ & Job Having Count(*) < 5

→ Write a Query to find out the number of clerks in Each department only if the count is greater than 1

Sol:- Select deptno, ~~Job~~ Count(*) From Emp Where Job = 'clerk'
Group By deptno Having ~~deptno~~ count(*) > 1

(or)

Select deptno, count(*) From emp
Where Job = 'clerk'
Group By deptno
Having count(*) > 1



Order by clause:- It is used for sorting the data either in ascending (or) descending order of a query basing on a specified list of column(s).

Syntax:- order By <collist> [Asc | Desc] (default is ascending only)

Eg:- Select Empno, ename, sal, comm, Deptno. From Emp order by sal

Select Empno, Ename, Sal, Comm, Deptno. From Emp order by sal Desc

Selec: Empno, Ename, Sal, Comm, Deptno From Emp where Deptno = 20 order By Sal, Comm

Note:- When we have multiple columns in order by clause, the data first gets arranged basing on the first column and if any duplicate values in the first column then it will take the support of second column for arrangement or else second column will not be used.

Eg:- Select Empno, Ename, Sal, Comm, Deptno From Emp
Where Deptno = 20
order By sal, comm desc.

(here one column in asc & other in desc)

2/07/2013 Tuesday

OFFSET AND FETCH options under order By clause.

When applying the order by clause using the offset option we can eliminate the number of rows from the starting record

Eg:- Select Empno, Ename, sal from Emp order by sal offset 5 Rows

- Skips first 5 records in the result

FETCH:

By using the fetch we can specify the number of rows to return after the offset clause is processed

Eg:- Select Empno, Ename, sal from Emp order by sal offset 5 Rows Fetch Next 5 Rows only.

- Gets the rows from 6-10 in the result skipping first 5 rows.

Eg:- Select Empno, Ename, sal from Emp order by sal offset 0 Rows Fetch Next 5 Rows only.

- Gets the first 5 rows of the result.

SUB-QUERIES:

A query inside another query is known as a Sub-query.

Syntax:- Select <collist> From <Tname> Where <conditional operator> (<Select stmt>)

In a sub-query, first the inner query executes and gives the result to the outer query for execution.

→ Write a query to get the details of the Employees who is earning the highest salary.

Sol:- Select * From Emp Where Sal = (select Max(Sal) From Emp)

→ Write a query to get the second highest Salary Employee details in the organization.

Sol:- Select * From Emp ~~Where Sal =~~

Where Sal =
(Select Max(Sal) From Emp Where Sal <
(Select Max(Sal) From Emp))

→ Write a query to get the third highest salary Employee details in the organization.

Sol:- Select * From Emp.
Where Sal =
(Select Max(Sal) From Emp Where Sal <(Select Max(Sal) From Emp Where Sal <(Select Max(Sal) From Emp)))

We can Nest the queries upto 32 only

Note: In the above context to get the ~~nth~~ highest Salary Employee details we are writting n+1 queries. but the Max number of queries permitted in the Nesting is only 32.

→ Write a query to get the details of Employees working in Sales department.

Sol:- Select dept * From Emp Where Deptno = (Select Deptno From Dept where Dname = 'Sales')

→ Write a query to get the list of Employees working in Chennai.

Sol:- Select * From Emp Where Deptno = (Select Deptno From Dept where Loc = 'chennai')

→ Write a query to get the list of employees who is earning more than highest salary of deptno 30.

Sol:- Select * From Emp Where sal >
(Select Max(Sal) From Emp where Deptno = 30).

(or)

Select * From Emp where sal > All
(Select Sal From Emp where Deptno = 30).

→ Write a query to get the details of the employee whose is earning less than the lowest salary of deptno 20.

Sol:- Select * From Emp where Sal < All
(Select sal From Emp Where Deptno = 20)

(or)

Select * From Emp Where sal <
(Select Min(Sal) From Emp where Deptno = 20)

→ Write a query to get the list of Employees whose is earning less than the highest salary of Deptno 10.

Sol:-

```
Select * From Emp Where
Where Sal < Any
(Select Sal From Emp Where Deptno = 10)
```

(or)

```
Select * From Emp
Where Sal <
(Select Max(Sal) From Emp where Deptno = 10)
```

(or)

```
Select * From Emp
Where Sal < Some
(Select sal From Emp Where Deptno = 10)
```

* All, Any & Some operators are used with sub-queries when the inner query is returning multiple values.

* All operator implies the condition should satisfy all the values in the list.

---

* Any (or) Some (both are same) operator should satisfy any value in the conditions list.

Any, Some and All operators can be used only in conjunction (or) association with a sub-query.

→ Write a query to get the list of Employees who is earning the highest salary in each department.

Eg:- Select * From Emp
Where Sal IN
(Select MAX(Sal) From Emp Group By Deptno)

IN operator also can be used with sub-queries when the inner query is returning more than 1 value. Just like we used ANY (or) Some and all operators. But, IN operator is used for Equality condition Whereas, ANY (or SOME AND ALL operators are used in case the condition is less than, less than or equal or greaterthan, greaterthanequal.

The above Query is First getting the highest salary of Each department and then using those values in the outer query. So, the problem in this execution is if the ~~values~~ salaries matches with other salaries in the department Even if they are not highest of the department will be retrived.

10	5000
10	4000
20	4000
30	4000

O/P

10	5000
10	4000
20	4000
30	4000

Eg: Select * From Emp where
Deptno+'/'+Sal IN
(Select Deptno + '/' + MAX(Sal) From Emp Group By Deptno)

To overcome these types of problems in oracle we are given with a concept of multicolumn. so that the Comparision is Performed with department & Salary also

Select * From Emp Where
(Deptno, Sal) IN
(Select Deptno, MAX(Sal) From Emp
Group By deptno) ;;

// Executes only in (Oracle)

This type of multicolumn comparisions are not supported in Sql server. So we need to adopt our own logic in getting the results.

Eg:- Select * From Emp Where
Deptno + sal IN (Select Deptno + Max(Sal) From Emp
Group By deptno).

→ Write a query to get the details of the Employee who is the most senior in the organization.

Eg:- Select * From Emp Where
Hiredate = (Select Min(Hiredate) From Emp)

→ Write a query to get the details of the most senior Employee in each department.

Sol:- Select * From Emp Where
Hiredate IN (Select Min(Hiredate) From Emp Group By deptno)

3-Jul-13 Wed

Co-Related Sub Queries:

Many queries can be evaluated by executing the sub query once and substituting the resulting values in to the Where Clause of the outer query. If queries that include a correlated Subquery (Repeating Subquery), the subquery depends on the outer query for its values. This means that the subquery is executed repeatedly. that might be selected by the outer query.

→ Write a query to get the details of the employee who is earning the highest salary in the organisation.

Sol:- Select * From Emp E Where 0 =
(Select count (sal) From Emp m
Where m.sal > E.sal)

In the above case 1st the outer query execute and then the inner query execute by comparing with the value outer query i.e it will count the no.of outer query salaries are greater than inner query salaries.

E

Empno	Sal
1001	5000
1002	4000
1003	3800
1004	4000
1005	3500
1006	3000

M

Empno	Sal
1001	5000 → 0
1002	4000 → 1
1003	3800 → 3
1004	4000 → 1
1005	3500 → 4
1006	3000 → 5

After performing a comparision it keep the record that matching with where condition which is nothing but the highest salary Emplyee details. so in this process if we use '1' in place of 0 it will give the 2nd highest salaried employee detail.

→ The Problem what we face in the above execution is if there are duplicate records ranking sequence will not be proper, i.e on the above case if we want to get the 3rd highest salary we can not use 2 because "2" is missing.

⇒ To overcome the problem rewrite the code as following.

```
Select * from Emp E Where 0 =
(
Select count (Distinct sal) From Emp M
Where m.sal > E.sal
)
```

E

Empno	sal
1001	5000
1002	4000
1003	3800
1004	4000
1005	3500
1006	3000

M

Empno	sal
1001	5000 → 0
1002	4000 → 1
1003	3800 → 2
1004	4000 → 1
1005	3500 → 3
1006	3000 → 4

→ Write a query to get the details of the department in which employees were working.

```
Select * From Dept Where Deptno In
(
Select Distinct Deptno From Emp
)
```

By using correlated sub query

```
Select * From Dept D Where Exists
(
Select Deptno From Emp E
Where E.Deptno = D.Deptno
)
```

Exists (<Stmts>)

Rows > 0 True
Rows = 0 False

Not Exists (<Stmts>)

Rows > 0 False
Rows = 0 True

→ Write a query to get the list of dept in which Employees are not working.

Sol:- Select * From Dept Where Deptno
Not IN
(Select Distinct Deptno From Emp)

Select * From Dept D Where Not exist
(
Select Deptno From Emp E Where
E.Deptno = D.Deptno
)

→ Write a query to get the list of employee who have sub-ordinate under them.

Sol:- Select * From Emp Where Empno In
(
Select Distinct Mgr From Emp)

Select * From Emp E Where Exists
(
Select * From Emp m where
m.mgr = E.Empno
)

Note:- Co-related subquery will be consuming more time to execute because every row of the inner query is getting compared with every row of the outer query. So, use a "correlated subquery" only if there is no chance are using a "Subquery".

JOINS

These are used for retriving the data from one or more tables at a time.

⇒ Joins can be used in any of the following ways.

⇒ Equi – Joins
⇒ non Equi – joins
⇒ Self join
⇒ Cartesion join
⇒ outer join

Equi – joins :- If two or more tables are combained using in Equality Condition we call it as a Equi join.

Q: Write a Query to get the Matching records from emp and DEPT tables.

Select E. EMPno, E. Ename, E. Job, E. Mgr, E. Hiredate
E. Sal, E. Comm, E. Deptno, D. Deptno, D. Dname,
~~D. loc~~ D. LOC From 'EMP E, Dept D. Where
E. Deptno : D. Deptno

In the above case it will retrive the records form the both the two tables only. If that the DEPTNO matches with the two tables, So it will not retrive the information of Dept no "40" from Dept table Because

a match is not available in Emp table

Note: The above Query will retrive the data from the both the two tables but the Syntax of joining the table is not a ANSI Standard. To write joins in ANSI Standard we are given with three option.

⇒ ~~Master~~
⇒ Inner join
⇒ Cross join
⇒ outer join

Inner join :- using "Inner join" we can implement Equi-join, non Equi-join as well as Self join also.

Equi-join in ANSI Standard:-

Select E.EmpNo, E.Ename, E.Job, E.mgn, E.Hiredate, E.Sal, E.Comm, E.Deptno, D.Deptno, D.Dname, D.Loc From EMP E Inner join (only) join Dept D on E.Deptno = D.Deptno

Note: In ANSI Standards to write a join Statement we explicitly use the join keywords between we explicitly the tables which gives descriptiveness and also to join the tables with the conditions we use the "on" keyword but not "where".

4/07/2019 Thursday

Loading data from multiple tables using Equi-Join Condition:-

By using an Equi-Join, we can combine the data of any number of tables provided all these tables contains a common column

DeptDetails

Did	Deptno	comments
1	10	
2	20	
3	30	
4	40	

→ Write a query to join the three tables Emp, Dept & DeptDetails

Sol:- Non-Ansi Join:

```
Select E.Empno, E.Ename, E.Job, E.mgr, E.Hiredate, E.Sal,
E.Comm, E.Deptno, D.Dname, D.Loc, DD.id, DD.Comments
From Emp E, Dept D, DeptDetails DD
Where E.Deptno = D.Deptno and
D.Deptno = DD.Deptno.
```

Ansi - Standard Join:

```
Select E.Empno, E.Ename, E.Job, E.mgr, E.Hiredate, E.Sal,
E.Comm, E.Deptno, D.Dname, D.Loc, DD.id, DD.comments
From Emp E
Inner Join Dept D on E.Deptno = D.Deptno
Inner Join DeptDetails DD on E.Deptno = DD.Deptno.
```

Non-Equi Join: If we join tables with any condition other than Equality condition we call it as a non-Equi Join.

Salgrade

Grade	Losal	Hisal
1	1300.00	1800.00
2	1801.00	2700.00
3	2701.00	3500.00
4	3501.00	5000.00
5	5001.00	8000.00

Write a query to get the data from Emp and Salgrade tables by displaying the grade of the employee basing on the salary.

Sol:- Non-Ansi Join:

Select E.Empno, E.Ename, E.Job, E.Sal, S.Grade, S.Losal, S.Hisal, From Emp E, Salgrade S where E.Sal Between S.Losal and S.Hisal.

Ansi standard Join:

select E.Empno, E.Ename, E.Job, E.Sal, S.Grade, S.Losal, S.Hisal from Emp E inner Join Salgrade S on E.Sal Between S.Losal and S.Hisal.

→ Write a Query to get the data from Emp, Dept and Salgrade tables.

Sol:- Select E.Empno, E.Ename, E.Job, E.Sal, D.Deptno, D.Dname, D.Loc, S.Grade, From Emp E inner Join Dept D on E.Deptno = D.Deptno.

Inner Join)x

Non-Ansi Join

Sol:- Select E.Empno, E.Ename, E.Job, E.Sal, D.Deptno, D.Dname, D.Loc, S.Grade, S.Losal, S.Hisal From Emp E, Dept D, Salgrade S Where E.Deptno = D.Deptno And E.Sal Between S.Losal and S.Hisal.

Ansi standard Join:

Select E.Empno E.Ename, E.Job, E.Sal, D.Deptno, D.Dname, D.Loc, S.Grade, S.Losal, S.Hisal From Empe Inner Join Dept D on E.Deptno = D.Deptno

Inner Join Salgrade on E.Sal Between S.Losal and S.Hisal

SELF Join:- Joining a table to itself for getting the results is a Self Join

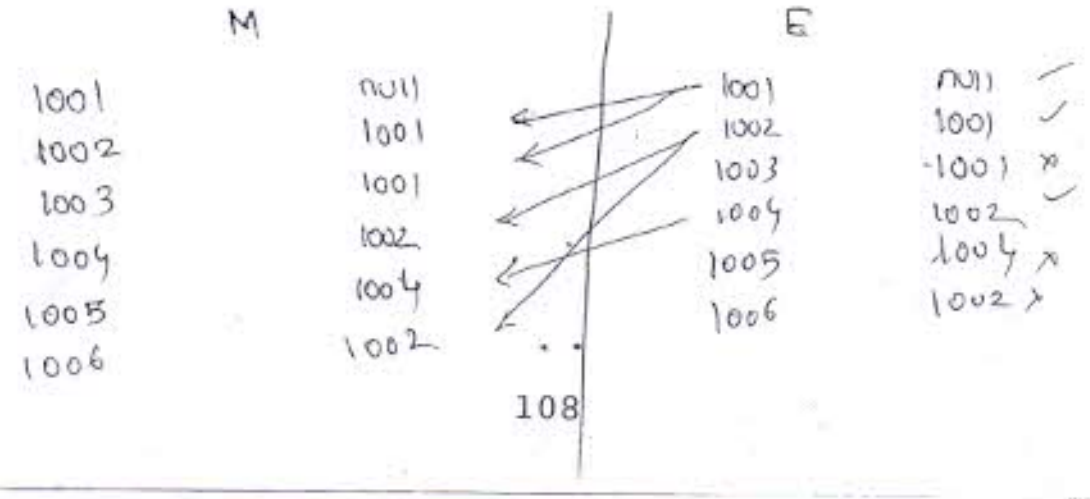
→ Write a query to get the details of the Employee who have Sub-c'dinates under them by using a Join.

Sol:- Non Ansi Join:

Select Distinct E.Empno, E.Ename, E.Job, E.Sal, E.Deptno, From Emp E, Emp M where E.Empno = M.Mgr

Ansi standard Join

Select Distinct E.Empno, E.Ename, E.Job, E.Sal, E.Deptno, From Emp E Inner Join Emp M on E.Empno = M.Mgr:



CARTESIAN (or) CROSS JOIN:

If two or more tables are combined with each other without any condition we call it as a cartesian (or) a cross Join. Here Each row of the first table goes and Joins with each row of the second table. So if the first table as m rows and the second table as n rows The output will be (mxn) rows (Cartesian product).

1. Non-Ansi Join:

```
Select * From Emp E, Dept D
```

2. Ansi standard Join

```
Select * From Emp E cross Join Dept D.
```

1. Non-Ansi Join

```
Select * From Emp E, Dept D, Dept-Details DD
```

2. Ansi standard Join

```
select * From Emp Cross Dept D Cross Join Dept Details DD.
```

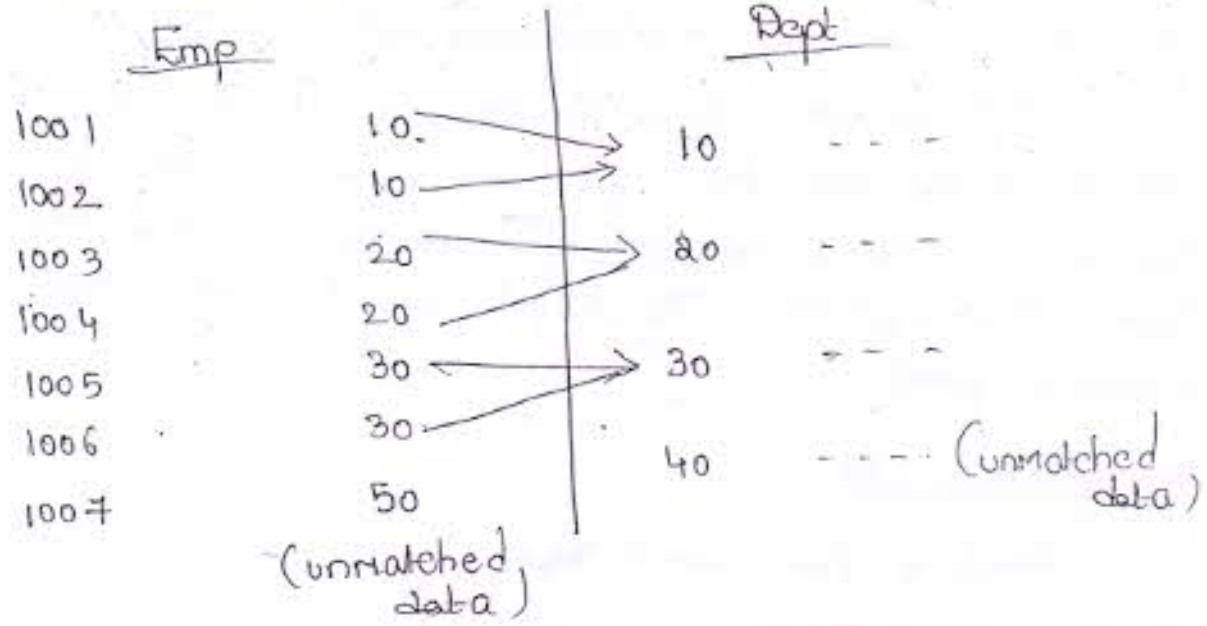
o/p (240)

OUTER JOIN:

It is an extension for the Equi-Join that is in an Equi-Join condition we will be getting the matching information from the tables used in the queries. Where as if a record(s) of a table doesn't have the matching data in the other table that record(s) will not be retervied.

In case of an outer Join just like our Equi-Join will retervive the matching data from all the tables as well as the unmatched data also. Where the unmatched data can be present in the L.H.S table (or) R.H.S table

(or) both tables also.



Outer Joins are of three types:

1. Left outer Join: Retrieves the matching data from both the tables as well as unmatched data from Left hand side table.
2. Right outer Join: Retrieves the matching data from both the tables as well as unmatched data from Right hand side table.
3. full outer Join: Retrieves the unmatched data from both the tables plus unmatched data from L.H.S tables plus unmatched data from R.H.S tables also.

Left outer Join:

Eg:- Select E.Empno, E.Ename, E.Job, E.Mgr, E.Hiredate, E.Sal, E.Comm, E.Deptno, D.Dname, D.Loc, From Emp E Left outer Join Dept D on E.Deptno = D.Deptno.

Right outer Join:

Eg: Select E.Empno, E.Ename, E.Job, Emgr, E.Hiredate, E.Sal, E.Comm, E.Deptno, D.Deptno, D.Dname, D.LOC From Emp E Right outer Dept D on E.Deptno = D.Deptno.

Full outer Join:

Select E.Empno, E.Ename, E.Job, E.mgr, E.Hiredate, E.Sal, E.Comm, E.Deptno, D.Deptno, D.Dname, D.Loc from Emp E Full outer Dept D on E.Deptno = D.Deptno.

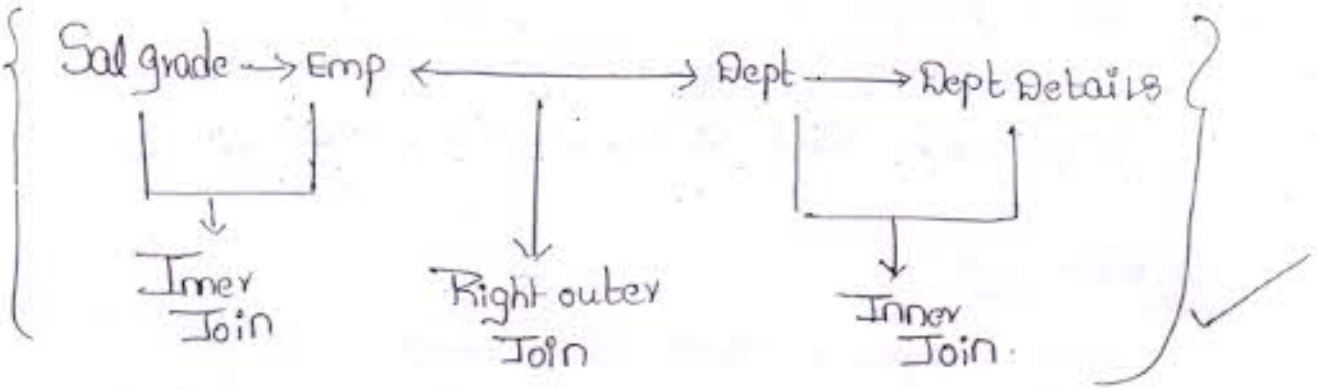
5/07/2013 Friday

→ Write a query to get a matching and unmatching data from the tables Emp, Dept, DeptDetails and (Salgrade.)

Sol:- Select E.Empno, E.Ename, E.Job, E.Sal, ~~E.Deptno~~, D.Deptno, D.Dname, D.Loc, DD.id, DD.comments, ~~S.Grade, S.Losal, S.Hisal~~ from Emp E Right outer Join Dept D on E.Deptno = D.Deptno. Inner Join DeptDetails DD on D.Deptno = DD.Deptno.

Note:- In the above query, we are not retriving the information from Salgrade table. If we want to get the data from that table also with matched and unmatched data the query must be as following.

Eg:- Select E.Empno, E.Ename, E.Job, E.Sal, S.Grade, D.Deptno, D.Dname, D.Loc, DD.id, DD.comments from Salgrade S Inner Join Emp E on E.Sal Between S.Losal and S.Hisal Right outer Join Dept D on E.Deptno = D.Deptno Inner Join DeptDetails DD on D.Deptno = DD.Deptno



{Emp → Dept → Dept Details → salgrade } ✗

SYNONYMS:- It is an object which can be created as an "alias" for any object like table, view, procedure, function etc.

Syntax:- Create Synonym <Name> for <object Name>

Eg:- Create Synonym E For Emp

Select * from E

SEQUENCE:- This is a new object introduced in Ser. Sql Server 2012 used for generating a sequence of numeric values according to the specifications with which the sequence is created.

Sequence is capable of generating numeric values in ascending and descending order also by providing a restart (or) cycle option. When it reaches the Max.. value.

Like Identity, sequences are not associated with specific tables.

Syntax:-

CREATE SEQUENCE <Name>

[AS [type]]

[Start with <Constant>]

[INCREMENT BY <Constant>]

[{MINVALUE [<constant>]} | {NO MINVALUE}]

[{MAXVALUE [<constant>]} | {NO MAXVALUE}]

[CYCLE | {NO CYCLE}]

[{CACHE [<constant>]} | {NO CACHE}]

Type refers to the type of value which should be generated by the sequence must be an integer type only. It allows the following types

tinyint - Range 1 Byte
Smallint - Range 2 Bytes
Int - Range 4 Bytes
Bigint - Range 8 Bytes (default)
Decimal and Numeric with a scale of 0.

Note:- Decimal and Numeric when used without a scale will be integer only.

START WITH: It is the first value written by the sequence. Start value must be a value lessthan or Equal to the MAXIMUM VALUE or greater than or Equal to the MINIMUM value of the Sequence. The default start value for a new sequence is Minimum value for ascending Sequence and Maximum value for descending Sequence.

INCREMENT BY: It is the value used for increment or decrement for the next value being generated by the sequence for each column.

If the increment is a negative value it is a descending sequence other wise it is ascending. The default increment is 1 and cannot be specified 0.

MINVALUE: Specifies the lower bound for the sequence object. The default min value for a new sequence is the minimum value of the datatype which is zero for the tinyint and Negative number for all other datatypes.

MAXVALUE: It specifies the upper bound FOR the sequence object. The default Max value for is the max value of the datatype.

CYCLE|(NOCYCLE): This property specifies whether the sequence object should restart or throw an exception when it is minimum or maximum value is exceeded. Default is Nocycle.

Note:- The cycling restart from the minimum value if it is incrementing sequence or maximum value if it is decrementing sequence but not from the start value.

(CACHE|(NOCACHE): This is used for increasing performance for applications that uses sequence object by minimizing the number of I/O's that are required to generate sequence Number.

If We Specify a Cache size, Sql Server will generabe that many number of values at a time and returns them one by one.

Note:- If we specify the cache Without cache Size, database Engine will select the size, however that value will not be consistent and not under our control.

When a Sequence is created with a cache option any unexpected Shutdown may result in the loss of Sequence number remaining in the cache.

Eg:- Create Sequence Myseq.
As Int
Start with 1001
Increment By 10

Select Next value for MySeq

Generating a value usi From the Sequence.

Syntax: Next value For <SeqName>

Eg:- Select Next value For Myseq

Eg:- Create Sequence Empseq
as int
Start with 2001
Increment by 1
Minvalue 2000
Maxvalue 2010
Cycle
Cache 5

Using Sequence in insert Statement:

```
Insert into Emp(Empno, Ename, Sal, Deptno)
  Values(Next value for Emp Seq, 'xxx', 3000, 40)
```

Altering a Sequence:- once the Sequence is created you have a chance of modifying the Sequence by changing any value we have specified but not, the data type of that Sequence.

Syntax:-

```
ALTER SEQUENCE <Name>
  [RESTART [WITH <constant>]]
  [INCREMENT BY <constant>]
  [{MINVALUE <constant>} | {NO MIN VALUE}]
  [{MAXVALUE <constant>} | {NO MAX VALUE}]
  [CYCLE | {NO CYCLE}]
  [{CACHE [<constant>]} | (NOCACHE}]
```

In an alter Sequence statement, we are provided with restart with option which is used for regenerating the values after altering the Sequence.

Eg:- Alter Sequence Empseq
  Restart with 2001
  Max value 2015.

9/07/2013 Tuesday

Creating a new table From Existing table:

If required we can create a new table basing on the columns of a single or multiple tables also as following.

Syntax:- Select <collist> Into <New Table> From <old Table> [Clauses]

Eg:- Select * into New_Emp From Emp

Select E.Empno, E.Ename, E.Job, E.Sal, D.Deptno, D.Dname, D.Loc, DD.Did, DD.Comments into EmpDetails From Empl
Inner Join Dept D on E.Deptno = D.Deptno
Inner Join DeptDetails DD on D.Deptno = DD.Deptno

Creating a blank table by copying the structure.

Select * into Blank_Emp From Emp Where 1 = 2
Select * From Blank_Emp

Using identity function while creating a new table from existing table.

Syntax:- Identity (data_type [, seed, increment]) as colname

Identity function can be used only in a select with an into table clause to insert an identity column into a new table.

Ex:- Select Identity (int, 1, 1) as Id, Empno, Ename, Sal, Comm, Sal + ISNULL (comm, 0) as TotalSal Into Sal_Emp from Emp.

VIEW: It also a database object, much like a table but Logical We can either call it as a logical or virtual table. Because it doesn't have any physical existance.

Syntax:- CREATE VIEW <Name> [(column [,...n])]
[WITH <view_attribute> [,...n]]
AS Select_statement [WITH CHECK OPTION]

<VIEW attribute>:
- ENCRYPTION
- SCHEMABINDING

When compared with table we have the following differences between a table and view.

1. Table is physical where as view is Logical.
2. Table is an independent object where as view is a dependent object that is a view depends on a table (or) tables from which it is loading the data.
3. When a new table is created from an existing table the new and old tables are independent of themselves that is the changes of one table will not be reflected into the other table where as if a view is created basing on a table any changes that are performed on the table reflects into the view and any changes performed on the view reflects into the table also.

Eg:- Create view Sales_view as select * From Emp Where Deptno = (select Deptno from Dept Where Dname = 'Sales').

Any Select Statement that can retreive information from a table (or) tables can be used for creating a view.

Eg:- Creating a view by specifying column names.

Create view SalDetails_View (Dno, Job, SalSum) As Select Deptno, Job, sum(Sal) From Emp Group By Deptno, Job

* We cannot use an 'order by' clause in a Select statement of Create view. But if there is a top clause in the Query order By is valid.

Eg:- Create view. Emp-Asc As select * From Emp order By Sal //Invalid

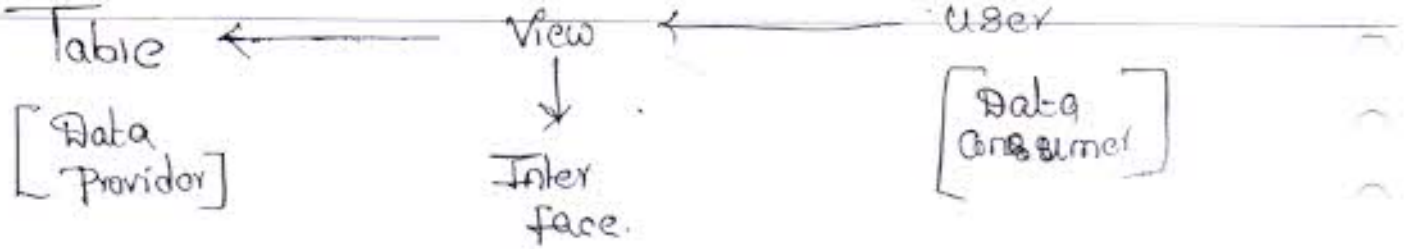
Create view Emp-Asc As Select Top 15 * From Emp order by Sal //valid.

Creating a view from multiple tables:

Create view Emp_Dept As Select E.Empno, E.Ename, E.Job, E.Sal, D.Deptno, D.Dname, D.Loc From Emp E Inner Join Dept D on E.Deptno = D.Deptno.

BENEFITS OF A VIEW:

1. When we want to provide access to a subset of data from the table we can then create a view with the required data and give permissions on that view so that, we can manage security.
2. When we want to access data from a table (or) tables by writing complex queries always in such cases if we create views basing on that queries we can directly query on the views only without writing complex queries Every time.
3. A view is a logical table, but what it stores internally is a select statement that is used for creating the view. So that, whenever a user performs any operations on the view like Select, insert, update or delete internally the view performs those operations on a table. Simply speaking it acts as a interface between the data provider (table) and the data consumer (user).



10/07/13 Wed

Metadata tables:- These all tables which stores Metadata (data about data) under them. All the Metadata tables are Predefined system tables where we can only query and retrive the data, from those table but can't Manuplate the data

(i) Sys objects:- This table contains the information of each and every object i.e created under that database. Like tables [System or users], views, constraints, sequences, synonym etc. We can find out what type of object it is we can make use of the "xtype" column of the table.

```
Select * from sysobject where xtype='u' --usertable
```

xtype values:-

S : System Table
U : User Table
V : View
PK : Primary key
UQ : unique
C : check
F : Foreign key
SO : Sequence
SN : Synonym
SP : Stored procedure
FN : Function

Syscolumn :-

This contains the Information of each and every Column i.e associated with each and every table or view

```
Select * from syscolumn where ID=object_id('Emp')--
```

-- Gets the columns of Emp table.

Syscomments:-

This contains the information of views and check constraints where we find a column text which contains the query used in the creation of a view on the condition i.e applied for a check constraints.

Eg:- Select *Text From Syscomments where Id=object_Id ('Sales_view') -- Gets the text associated with Sales view

→ View classification:-

A) i) Simple views and ii) Complex views

B) i) Updatable views and ii) Non-updatable views

1) SIMPLE VIEW & COMPLEX VIEW:

⇒ A view which is created basing on the columns of a single table is known as a simple table

Eg:- 1) Create view view1 as select Empno, Ename, Job, Sal, Deptno From Emp

2) Create View view2 as select * From Dept.

→ A view i.e created basing on multiple tables columns is a complex view.

Eg:- Create view View3 as Select E.Empno, E.Ename, E.Sal, S.Grade, S.Losal, S.Hisal From Emp E InnerJoin Salgrade S on E.Sal Between S.Losal and S.Hisal

→ A view which is created basing on a single table will also be considered as a Complex view provided If the query contain any of the following:

1) Distinct, Aggregate function, Group By clause, Having clause, calculated columns and set operators

Eg 1) Create view view4 as select Deptno, sum(sal) as salsum from Emp GroupBy Deptno

sum(sal) → Complex view

GroupBy → complex view.

2) Create view view5 as
select job from Emp where Deptno=20
Intersect
Select job from Emp where Deptno=30

→ UPDATABLE AND NON-UPDATABLE VIEW:

A view which allows manipulation to the data associated with the view we called it as a updatable view and if it doesn't allow manipulations it's a non-updatable view.

→ By default all the columns of a simple view table are updatable and complex views tables are Non-updatable

Note: We can update complex views based on multiple tables but not all the columns we can update only columns associated with one table.

With check option:- If at all a view is created by using a where condition later on if any manipulation are performed on that view against the where condition still those changes are accepted. To test this create a view as following.

Eg:- Create view Finance_view AS
Select * From Emp where Deptno=30

In the above case the view is created associating with a viewhere condition "deptno=30" but if we still trying to insert a record that violates our where condition that operation gets performed i.e

Insert Into Finance_view (Empno, Ename, Job, sal, Deptno)
values (1016, 'Abc', 'manager', 4500, 40)

→ If we want to restrict any DML operations on the view against the where condition we need to add the "with check option" either at the time of creating the view or we can add that option by Altering the view.

Eg:- Alter view Finance_view As select * From Emp
Where Deptno=30 With check option.

Insert Into Finance_view (Empno, Ename, Job, sal, Deptno)
values (1017, 'ABC', 'clerk', 2000, 40)

Now the above statement execution fail as the where condition of the view is violating.

With Encryption:- After creating a view we can checkout the statement we have return for creating the view under that the text column of syscomment table. But for any security reason if we don't want to disclosed that information to anyone we can hide it by using "with Encryption" option either while creating the view or Altering the view.

Eg:- Alter view Finance_view
With Encryption
As
Select * From Emp where Deptno=30
With checkoption)

→ Now if we verify text column under the syscomments tables it displays Null in the place of create view statement.

Eg:- Select Text From sysComments
Where id = object_id ('Finance_view')

Q) can we drop a table that has dependent views on it?

Ans:- yes we can drop a table even if any dependent views are associated with it, but the views are associated with it, but the views will not be droped. They will be still existing in the database only with the status as inactive object and all those views become active and start Functioning Provided the table is recreated.

Thursday
11 | 07 | 2013

WITH SCHEMABINDING:

If a view is created by using the attribute "schemabinding" we "cannot drop" on "alter the table columns" on which the view is dependent.

Eg:- Create view marketing_Emp
With schemaBinding
AS Select Empno, Ename, Job, Sal, Deptno
From dbo.Emp Where Deptno = 10

→ After creating the view we cannot drop the Emp table or alter any columns that are specified in the view.

When we want to use the schemabinding option it is must to specify each and every column name in the select statements but cannot use "*".

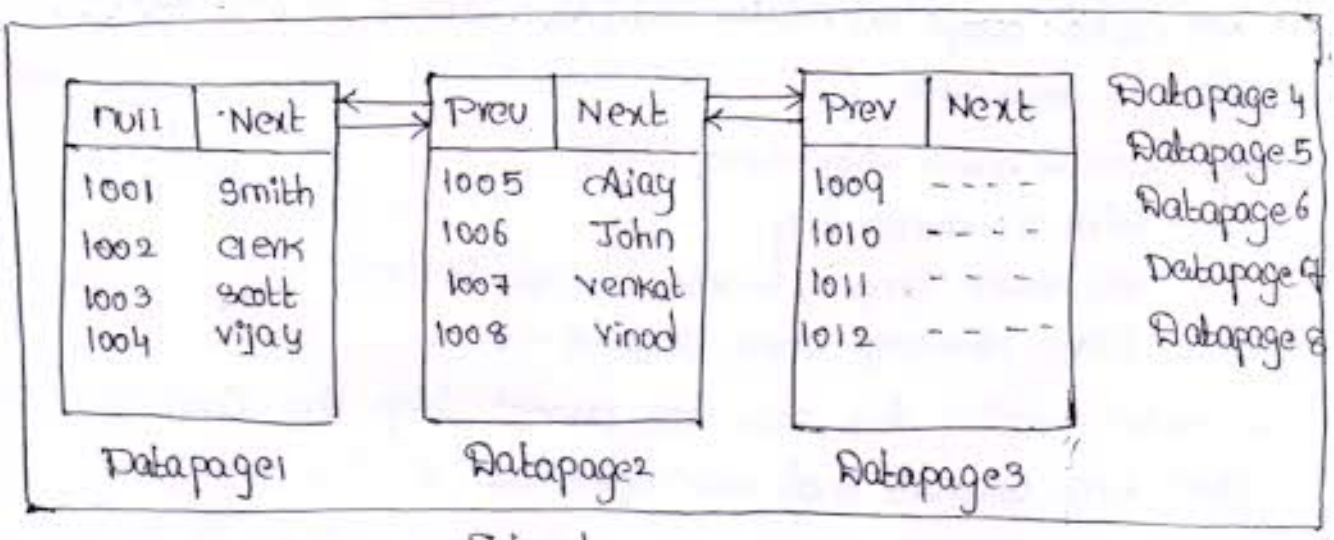
→ While using the schema binding option the table name must be prefix with the owner name is dbo. which tells the current user is only the owner of the table.

Note:- If required we can use the with Encryption and with schemaBinding option at the same time.

Eg:- Alter View Marketing_Emp
With Encryption, schemaBinding
As Select Empno, Ename, Job, sal, Deptno
From dbo. Emp where Deptno = 10

How Data Is Stored under a DataBase?

Sql Server Stores data in it under data pages, where a datapage is a minute location for storing of the information. A datapage will be having a size of 8KB and every 8 data Pages are stored under a logical container known as an "extend"



Extend

⇒ If all the datapage in an extend are filled for storing new data Sql Server creates a new extend with 8 datapages at a time.

Q How will the database engine retrives the information from a table:

Whenever the database engine want to retrive the information from the table it will adopt two different mechanism for Searching the data.

→ Full page scan
→ Index scan

→ In the first case sql server will search for the required information in each and every page to collect the information. So, if the table has more rows it will be taking lot of time for scanning all the data. so it is a time consuming process.

→ In the second case sql server without searching into each and every datapage for retriving the information it will make use of an index for retriving the information, where an index is a pointer to the information what we are retrive which can reduce the disk I/o operation saving the time, But if we want to use index scan for searching the data first the index has to be created.

Syntax:- Create [unique] [clustered / Non-clustered] index <index Name> on <Table Name> (<colist>)

Note:- Whenever an index is created on a column or columns of a table internally an index table gets created maintaining the information of a column on which the index is created as well as address (pointer to the row corresponding to a column)

Index Table

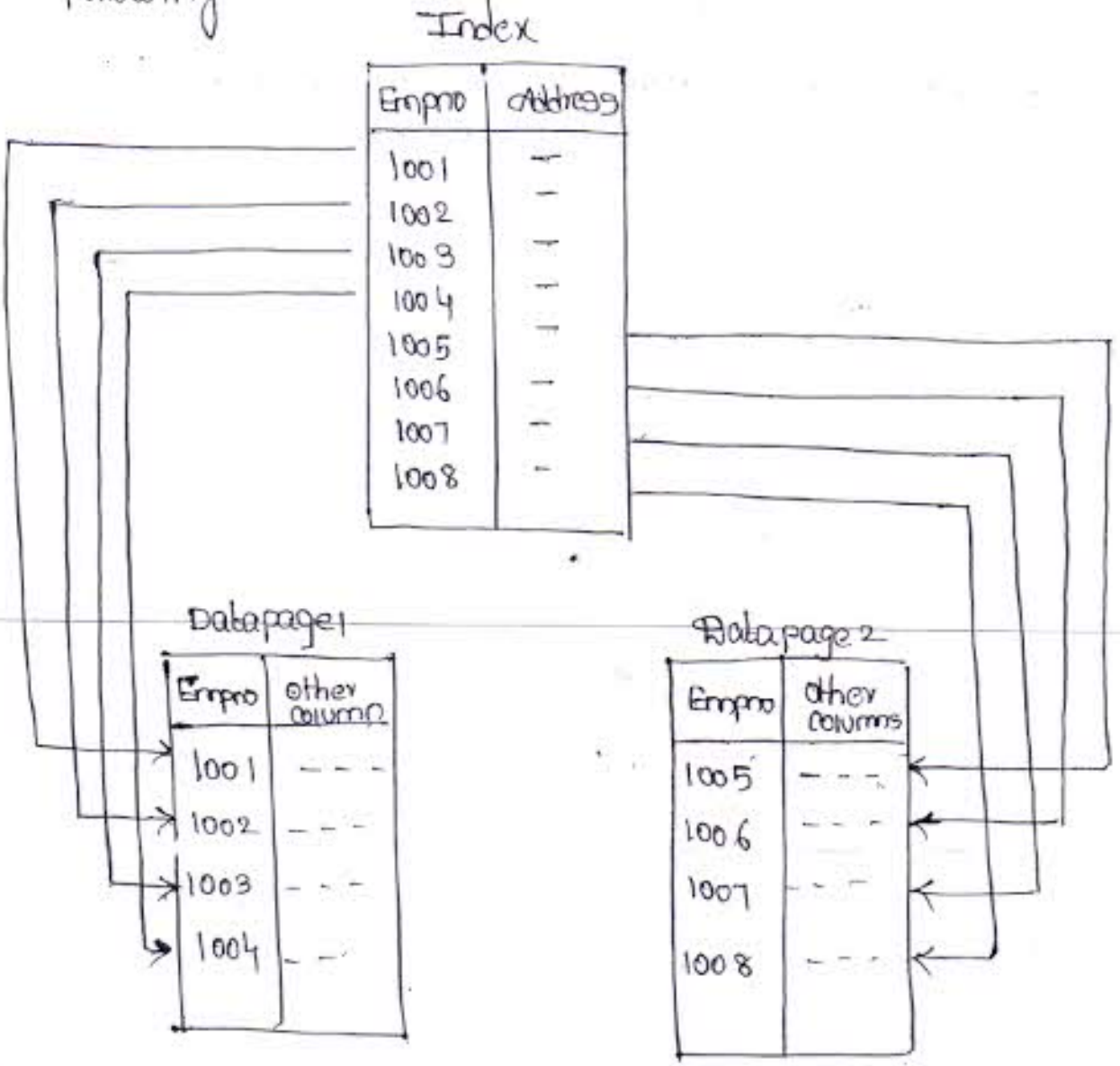
<colist>	Address

CLUSTERED INDEX:- In this case the arrangement of the data in the index table will be same as arrangement of the data of actual table.

Eg:- The index we find in the start of a book.

Note:- Indexes will arrange the information under them by adopting a structure called B-tree structure.

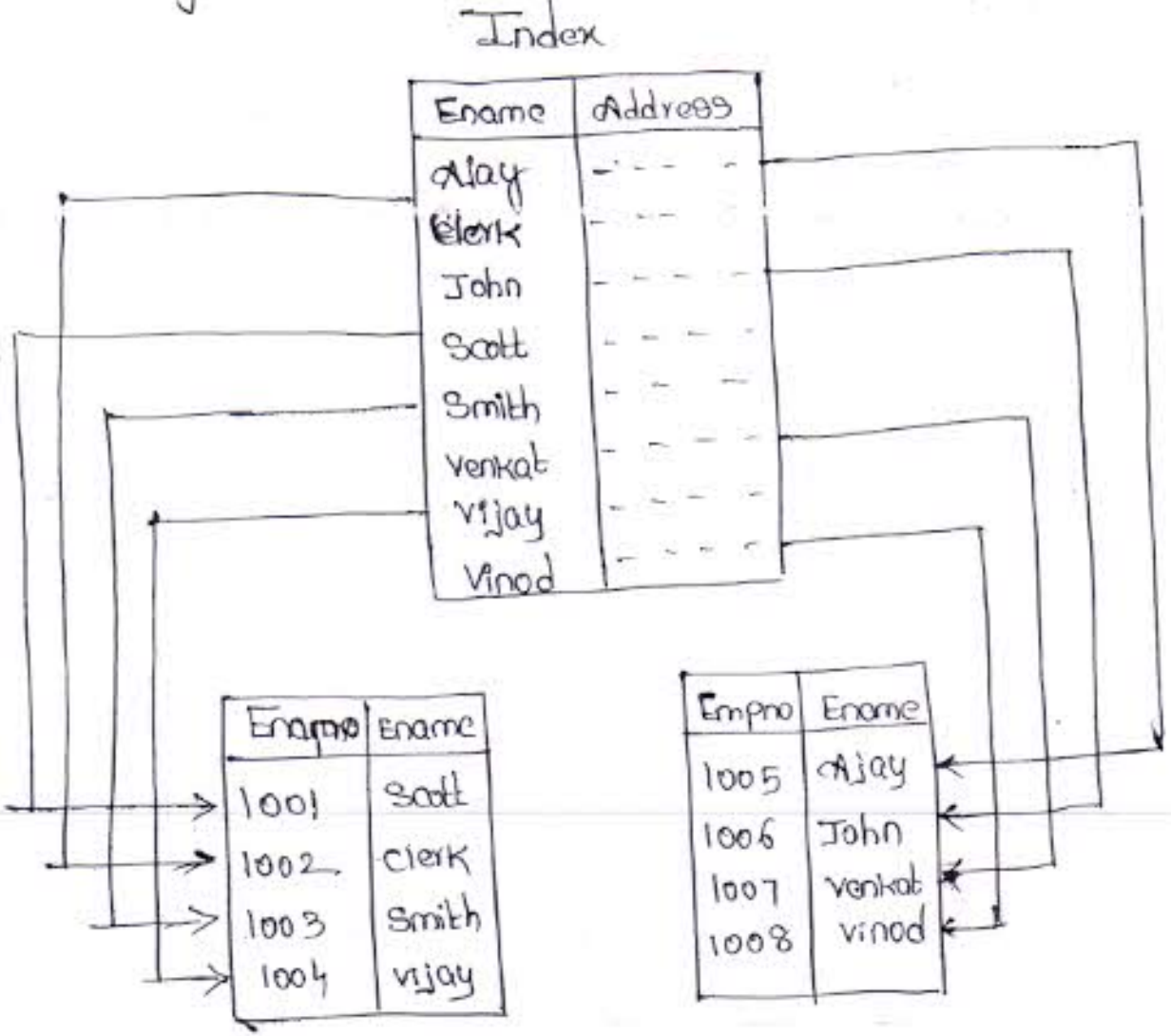
→ Suppose if a clustered index is created on the Employee no column of Emp table internally we will find the information as following.



NON CLUSTERED INDEX: In this case the arrangement of data in the index page or table will be different from the arrangement of in actual table.

Eg:- The index we find in the end of the book.

Suppose a nonclustered index is created on Ename column of Emp table the arrangement of the data will be as following.



Unique Index:- If the index is created by using "unique" option that column on which the index is created will not allow duplicate values i.e it works as a "unique constraints". Unique constraint can be either unique clustered or unique non-clustered also.

Note:- While creating an index if clustered or non-clustered is not specified. Default is non-clustered.

Q:- How many Indexes a table can have?

We can create a maximum of 250 indexes in which only one can be clustered and remaining all are nonclustered

Are indexes created Implicitly or should be create them Explicitly.

When ever we impose a primary key constraint on a tables column internally a "unique clustered" index gets created. Where as if a unique constraint is impose on any column internally a "unique nonclustered" index gets created.

→ Creating indexes on Emptable:-

As the Empno column is imposed with a pk it will contain a unique clustered index implicitly Apart from that we can create any number of nonclustered indexes on the table as following.

Eg: create Nonclustered index Ename_ind on Emp(Ename)
create index Sal_ind on Emp(Sal)

Eg:- Create table Student (Sid int, Sname varchar(50))
Create unique clustered index Sid_Ind on Student (Sid)

Note:- In the above case the index will not provide the functionality of primary key for Sid column but still we will get the functionality of unique constraint.

12-01-2013 Friday

When SQL Server uses Indexes:-

Sql server uses indexes of a table provided the select or update or delete statement contained "where" condition in them and moreover the where condition column must be a indexed column.

If the select statement contain an "order by" clause also indexes will use.

Note:- when sql server is searching for information under the database first it verifies the best execution plan for retriving the data and uses that plan which can be either a full page scan or an index scan also.

Eg:- Select * From Emp X
Select * From Emp Where Empno = 1003 ✓
Select * From Emp where Ename = 'scott' ✓
Select * From Emp Where Sal > 3000 ✓
Select * From Emp where Soundex(Job) = 'manager' ✗
Select * From Emp Where Soundex (Ename) = Soundex ('Smyth') ✗
Select * From Emp order by Sal Desc

When Should we create Indexes on a Table:

We need to create index on a table columns provided those columns are frequently used in "where condition" or "order By clause".

-> If is not adviced creating an index on each and every column because more number of indexes can degrade the performance of database also because every modification we make on the data should be reflected into all the index tables.

Transaction Control Language :- [TCL]

A Transaction is a unit of work or set of statement (Insert, update & delete) which should be executed as one unit.

The rule of transaction tells that either all the statements in the transaction should be execution successfully or non of those statement to be executed.

To manage transaction we have provided with transaction control language, that provides a commands like "commit Transaction", "Rollback Transaction", "save Transaction.

Commit Transaction:- Marks the end of a successful transaction which will make all data modifications performed since the start of the transaction permanent under the table and frees the resources held by the transaction.

Rollback Transaction:- It will bring back an impicit or explicit transaction to the begining of the transaction or erase all data modification made from resources held by the transaction

Sql server provides or supports 3 different modes of transaction

1) Auto commit transaction mode [default]
2) Implicit Transaction mode
3) Explicit Transaction mode.

Auto commit Transaction:-

In this mode programmers are not responsible for begining a transaction or ending a transaction when ever we perform or execute any DML statement - sql server only will Begin a transaction and ends the transaction with a commit or Rollback. So, programmers doesn't have any control on them.

Implicit transaction mode:-

In this mode Sql server will begin a transaction implicitly before execution any DML Statement and developers are responsibility to end the transaction with a Commit or Rollback once a transaction ends automatically a new transaction start.

→ To use implicit transaction mode need to explicitly execute the following command.

Set Implicit_Transaction (on/off)

If the property value is set as on it sets the connection into implicit transaction mode and if it is off returns the connection to auto commit transaction mode

```
Set Implicit_Transaction ON
update Emp set sal = 5000 where Empno=1002
Commit or Rollback
update Emp set sal = 6000 where Empno=1002
Commit or Rollback
```

Explicit Transaction Mode:-

In this mode programmer is only responsible for beginning the transaction and ending the transaction. To begin a transaction we can use "Begin Transaction" statement and End it either with a "Commit or Rollback"

Eg:-
```
Begin Transaction
    update Emp set sal = 6000 where Empno=1001
    Commit or Rollback
```

Begin Transaction

Update Emp set sal = 6000 where Empno = 1002

Commit or Rollback.

13/07/2013 Saturday

SAVE TRANSACTION: This is used for dividing (or) breaking a transaction into multiple units. So that we will have a chance of rollbacking a transaction upto certain a location.

When a user sets a save point within a transaction the save point defines a location to which a transaction can return if part of the transaction is conditionally cancelled.

If a transaction is rolledback to a savepoint, it must proceed to completion of the transaction with a commit statement (or) rollback statement.

Code:

```
Begin Transaction
update Emp Set Sal = 5500 Where Empno = 1001
update Emp Set Sal = 5000 where Empno = 1002
Save Transaction S1
update Emp set Sal = 4500 Where Empno = 1003
update Emp set Sal = 4000 Where Empno = 1004
Save Transaction S2
update Emp Set Sal = 3500 Where Empno = 1005
update Emp Set Sal = 3000 Where Empno = 1006
```

In the above case we are breaking the transaction into three units. So we have a chance of rollbacking the transaction either completely so that all the six statements gets rollbacked (or) rollback upto the save point S_1. So, that the last 4 statements gets ronbacked and then commit the first 2 (or) rollback upto savepoint S_2 and commit the first 3 4.

To end the above transaction we can use any of the following statement.

Rollback -- Will end the transaction.

(or)

Rollback Transaction s_1 -- Will not end the transaction

Commit -- Will end the transaction.

(or)

Rollback Transaction s_2 -- Will not end the transaction

Commit - Will end the transaction.

We have a chance of rollbacking upto a savepoint but can never commit upto a savepoint.

INDEXED VIEWS: As we are aware that a view is a logical table, that is will not store any data in it. What it stores is only the select statement used for creating the view. So that whenever we query on a view, the view will internally query the data from the table and present us. But if the view contains any calculated columns in it are aggregate columns in it whenever we query on the view all the calculations and aggregations has to be performed every time. So if there are more number of rows under a table the process will be time consuming. To overcome this problem we can make the view as physical by defining a unique clustered index on that view.

Creating a unique clustered index on a view improves query performance of a view because now the view is stored in database in the same way a table is stored. That is it becomes Physical

If we want to create a indexed view we have the following restrictions:

→ The view must be a schemabinded view
→ The view should compulsory Count_Big aggregate function in it.
→ If the aggregate functions are associated with any columns if at all column allows null value in it we should associate it with ISNULL function.

Ex:- Create View SalDetails
With SchemaBinding As
Select Deptno, Count_Big(*) As Empcount,
Sum (ISNULL(Sal,0)) As SalSum
From Dbo.Emp Group By Deptno.

As of now, the view is Logical to make it physical we need to define a index on the view as following.

Ex:- Create unique clustered index SalDetails_Ind
on SalDetails (Deptno).

Note: In the above case after defining the index view becomes physical, but any modifications performed on the table reflects into the view.

DROPPING AN INDEX: We can drop an index that is created on a table or view as following.

Syntax:- Drop Index <Name> on <TName>

Drop Index Ename_ind on Emp — Ename drop cannot
Drop Index SalDetails_ind on SalDetails → on view we can drop