DATABASE MANAGEMENT SYSTEM

Data Base Management System is defined as collection of Software Program's used to store the data, manage the data, and retrieve the data. Data retrieval is based on three points

- Accuracy – Data Free from Errors.
- Timelines – With in a time
- Relevancy – Related Data

Maintaining of the data can be done in two ways.

- File System
- Data Base

Why Database Management System and why not flat files to maintain the data.
Flat Files has many disadvantages they are

- No Security
- Difficulty in accessing the data
- Redundancy
- Data Inconsistency
- Sequential Search Technique (Time Consuming Process)
- Indexing is not present in file system
- No Identifier (Related Information)

DATABASE: It is defined as Logical Container which contains related objects, Objects Includes Tables, Views, Synonyms, Stored Procedures, Functions, Triggers Etc.

MODELS OF DATABASE MANAGEMENT SYSTEM

- Entity Relationship Model
- Object Relation Ship Model
- Record Based Model
    - Hierarchical DBMS
    - Network DBMS
    - Relational DBMS
    - Object Relational DBMS
- Info Logical Model
- Semantic Based Model

****SQL Server 2005(9.0) & 2008 (10.0) Includes both RDBMS & ORDBMS Model.**

RDBMS = R + DBMS {R Stands for Referential Integrity Constraint}
Relation can be defined as Referential Integrity Constraint, with which we create relationships One to One & One to Many (Direct Implementation) Many to Many (Indirect Implementation)

DR EF CODD has outlined 12 rules often called as CODD'S rules if any **Database Software Satisfies 8 or 8.5 rules**, it can be considered as pure RDBMS Database.

- SQL Server Satisfies 10 CODD's Rules
- Oracle satisfies all 12 CODD's Rules
- MySQL Satisfies

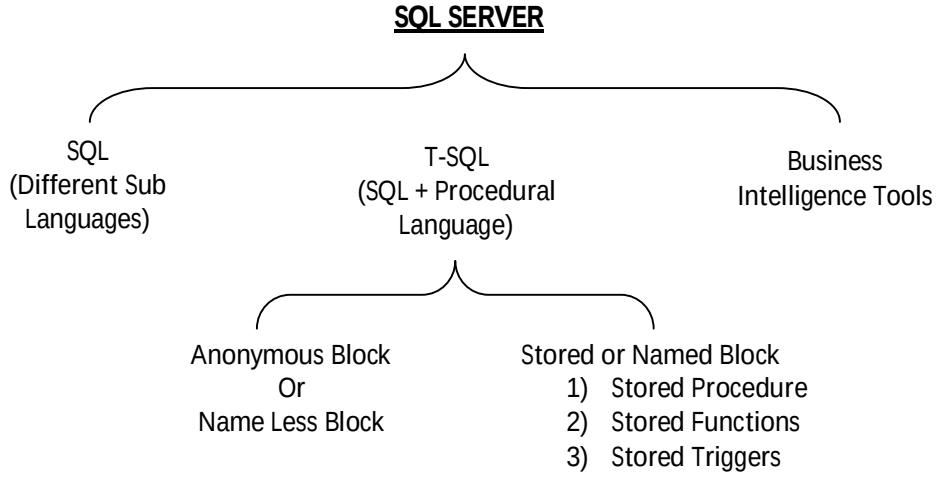
- Sybase (Teradata)
- MS Access is not a pure DBMS, because it satisfies only 6 rules, so it is called as Partial / Semi Database.
- RDBMS always presents the data in 2 Dimensional formats, i.e. in Rows and Columns, further to tables.
- Collection of characters "or" we call as IDENTIFIERS /FIELD/ ATTRIBUTE.
- Collection of columns: ROW or RECORD or TUPLE.
- Collection of Rows & Columns: Table or Entity or Objects or Relations.
- RDBMS always represent the data in a Normalized way.

COMPONENETS OF RDBMS:

- Collection of Objects or Relations to store data.
- Set of operators are applied on one or more relations that always produces a new relation.
- Data Accuracy – Data Consistency, Data Integrity.

SQL SERVER:

- It was introduced in the year 1989 by MICROSOFT as SQL SERVER 1.0
- It is GUI based RDBMS.
- It is CLIENT/ SERVER technology.
- It is associated with DATABASE ENGINE (set of compilers)
- It can be installed on specific Operating System (WINDOWS)
- Where as CLIENTS can be installed on any Operation System.
- SQL SERVER 2005 & 2008 are consisting of RDBMS & ORDBMS.



Differences between Oracle & SQL Server

ORACLE	SQL SERVER
• It was Introduced In 1979	• In 1989 It was Introduced
• It Is CUI Based RDBMS	• It Is GUI Based RDBMS
• It can be Installed on any	• It Requires Specific OS

Platform	(Windows)
• Provides High Security	• It Is Less Security
• It Is High Priced RDBMS	• It Is Low Priced RDBMS

VERSIONS OF SQL SERVER

SQL SERVER 1.0 ---------- 1989
SQL SERVER 4.0 ---------- 1993-95
SQL SERVER 4.5 ---------- 1993-95
SQL SERVER 6.0 ---------- 1995-97
SQL SERVER 6.5 ---------- 1995-97
SQL SERVER 7.0 ---------- 1998
SQL SERVER 7.0 ---------- 2000
SQL SERVER 9.0 ---------- 2005
SQL SERVER 10.0 ---------- 2008

FEATURES OF SQL SERVER

- GUI based RDBMS(Commands, Wizards)
- Provides High Security
- Provides High Scalability
- Provides High Availability
- Provides High Performance
- Supports XML

NEW FEATURES INTRODUCED IN SQL SERVER 2005:

XML Integration

- XML data type is introduced
- XML indexes are introduced
- FOR XML, OPEN XML
- QUERY (), EXIST ()

CLR INTEGRATION (Common Language Runtime Associated with Database Engine)

T-SQL ENHANCEMENTS

- DDL Triggers {Used To Perform Administration Related Tasks}
- LOGON Triggers.
- Structured Exception Handling [TRY, CATCH]
- Ranking Functions (Window Functions)
- Table Partitioning.
- Synonym as Database Object Is Introduced.
- Recursive Queries.

NEW TOOLS

- SQL SERVER Management Studio
- Query Analyzer (Command Based)
- Enterprise Manager (GUI Based)
- Query Editor (Command Based)
- Object Explorer (GUI Based)

CONFIGURATION TOOLS

- Service Management
- Client Network Utility
- Server

BIDS (BUSSINESS INTELLIGENCE DEVELOPMENT STUDIO)

- SSIS (SQL SERVER INEGRATION SERVICES)
- SSAS (SQL SERVER ANALYTICAL SERVICES)
- SSRS (SQL SERVER REPORTING SERVICES)
- SSNS (SQL SERVER NOTIFICATION SERVICES)

SQL SERVER NEW FEATURES IN 2008

- Intellisence For Query Editing
- Multi Server Query
- Query Editor Regions
- Object Explorer Enhancements
- Activity Monitors

T-SQL ENHANCEMENTS: SQL SERVER 2008 introduces several important new T-SQL Programmability features and enhancements to some existing ones.

Following are the key Features:

- Declaring and Initializing Variables
- Compound Assignment Operators
- Table Value Constructors(Support Through Values Clause)
- Enhancements to Convert Functions
- New Date and Time Data types & Functions
- Large UDTS(User Defined Types)
- Hierarchyid Data Type
- Table Type and Table Valued Parameters
- The Merge Statement (Grouping Sets Enhancements)
- DDL Trigger Enhancements
- Sparse Columns
- Filtered Indexes
- Large CLR User Defined Aggregates
- Multi Input CLR User Defined Aggregates
- The Order Option For CLR Table Valued Functions
- Object Dependencies
- Change Data Capture

EDITIONS OF SQL SERVER:

- ENTERPRISE EDITION – to perform administration related tasks
- STANDARD EDITION - to perform administration related tasks
- DEVELOPERS EDITION
- WORKGROUP EDITION
- EXPRESS EDITION

- MOBILE EDITION

SQL SERVER MANAGEMENT STUDIO

- It is used to perform command based operations and GUI based operations over a database.
- It consists of 2 sub components
  OBJECT EXPLORER – GUI + ADMINISRATION RELATED TASKS
  QUERY EDITIOR – COMMAND BASED OPERATIONS (SESSION)

Invoking SQL Server Management Studio through Run Command

SQL 2005 Click on Start and Run Type "SQLWB" (WB – WORK BENCH)
SQL 2008 Click on Start and Run Type "SSMS" (MS – MANAGEMENT STUDIO)

TYPES OF DATABASES

- System Data Base – Master, Model, MSDN, TempDB
- Sample Data Base – 2000(Pubs, Northwind), 2005(Adventures, Adventures DW)
- User Defined Data Base – Created by Developers

CREATING A USER DEFINED DATABASE:

- Database size in SQL SERVER 2000 was limited – 1048516TeraByte
- Database size in SQL SERVER 2005 was unlimited.
- Every database in SQL will be associated with two files.
- DATA FILES (STORES DATA)
  - Primary Data File (.MDF Master Data File)
  - Secondary Data File (.NDF Next Data File)
- LOG FILES (.LDF Stores Transaction Details)
- Every database will have at least one primary data file
- Tables will have logical data (ROWS & COLUMNS)
- Primary files have physical data (BLOCKS)
- Each block size will be of 8KB.
- A Database can be connected 32000+ NDF files.
- MDF will hold references of NDF files.
- Having more NDF files will help to make backup process easy.
- A Database can be associated by more than one .NDF & .LDF files.
- The advantage of creating NDF files is to have easy backup of database

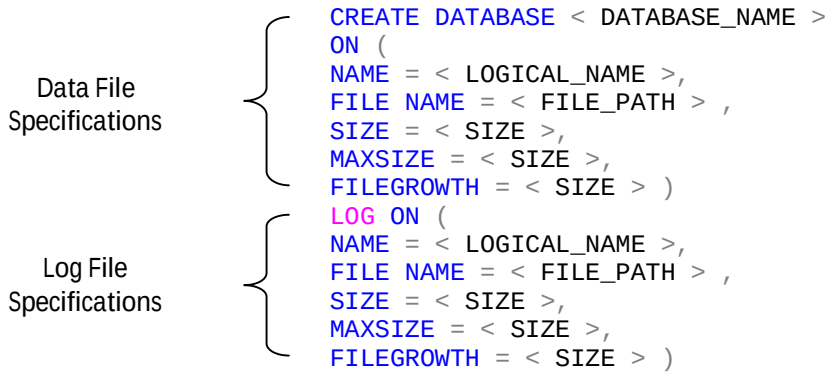
CREATING A DATABASE THROUGH – GUI

Step 1: Click on Object Explorer
Step 2: Click on Databases
Step 3: Right click on Database
Step 4: Click on to Create a New Database
Step 5: Enter DATABASE Name and Click Ok
Step 6: Specify the Size as .MDF Initial Size – 2MB Increment – 1MB MAX Size Unlimited
Step 7: Specify the Size as .NDF Initial Size – 1MB Increment – 10% MAX Size Unlimited

CREATING A DATABASE THROUGH QUERY/COMMAND

As Database creation is an administration related task, it should be carried out through MASTER system data base,

Syntax:



Example:

```
CREATE DATABASE TESTDB
ON (
NAME = TESTDB ,
FILENAME = 'C:\SQL Server\MSSQL.1\MSSQL\DATA\TESTDB.MDF' ,
SIZE = 5MB , MAXSIZE = 10MB , FILEGROWTH = 2MB )
LOG ON (
NAME = TESTDBLOG ,
FILENAME = 'C:\SQL Server\MSSQL.1\MSSQL\DATA\TESTDBLOG.LDF' ,
SIZE = 3MB, MAXSIZE = 8MB, FILEGROWTH = 2MB)
```

MODIFYING EXISTING DATABASE

Existing Database can be modified by using **ALTER DATABASE** Command, to perform different tasks like creating secondary files, changing sizes of existing files, removing the files and renaming a database.

Syntax:

```
ALTER DATABASE < DATABASE_NAME >
ADD FILE = < FILE_SPECIFICATIONS >
ADD LOG FILE  = < FILE_SPECIFICATIONS >
MODIFY FILE = < FILE_SPECIFICATIONS >
REMOVE FILE= < LOGICAL_NAME >
MODIFY NAME = < NEW_DATABASE_NAME >
```

Adding a new NDF File to an existing Database

```
ALTER DATABASE TESTDB
ADD FILE (
NAME = TESTDB1,
FILENAME ='C:\SQL Server\MSSQL.1\MSSQL\DATA\TESTDB1.NDF' ,
SIZE = 3 MB, MAXSIZE = 7MB, FILEGROWTH = 2MB )
```

Adding a new LDF file to an existing Data Base

```
ALTER DATABASE TESTDB
ADD LOG FILE (
NAME = TESTDBLOG1,
FILENAME ='C:\SQL Server\MSSQL.1\MSSQL\DATA\TESTDBLOG1.LDF' ,
SIZE = 3 MB, MAXSIZE = 7MB, FILEGROWTH = 2MB )
```

Modifying Existing Data & Log Files:

```
ALTER DATABASE TESTDB
MODIFY FILE (
NAME = TESTDB1, -- Logical Name
SIZE = 10MB, MAXSIZE = 12 MB )
```

Removing Files:

```
ALTER DATABASE TESTDB
REMOVE FILE TESTDB1 -- Logical Name
```

Renaming a Data Base:

```
ALTER DATABASE TESTDB
MODIFY NAME = DBTEST
```

Syntax:

```
SP_RENAMEDB  < OLD_DATABASE_NAME >,< NEW_DATABASE_NAME >
SP_RENAMEDB 'TESTDB','DBTEST'
```

Dropping a Database:

Syntax:

```
DROP DATABASE < DATABASE_NAME >
DROP DATABASE TESTDB
```

Command to Use Navigate between Data Base's

Syntax:

```
USE < DATABASE_NAME >
USE TESTDB
```

Command to get the Current Database Name - `SELECT DB_NAME()`

****SQL SERVER provides 1788 objects for every Database**

`SP_HELP:` It will return list of predefined objects provided by MASTER Data Base to a User Defined Database.

`SP_HELPDB`**:** This predefined Stored Procedure will provide list of database's available in SQL SERVER.

`SP_HELPDB <DATABASE_NAME>` - It will provide the complete information of the database.

STRUCTURE QUERY LANGUAGE – SQL

SQL is a language used to communicate with DATABASE SERVER.
Initially when SQL was introduced it was by name 'SQUARE' this language was modified and was released by a new name 'SEQUEL' , later was renamed to SQL by removing all vowel's in the SEQUEL.

- SQUARE – SPECIFICATIONS OF QUERY AS RELATIONAL EXPRESSIONS
- SEQUEL – STRUCTURED ENGLISH QUERY LANGUAGE
- SQL – STURCTURED QUERY LANGUAGE

SQL is a product of IBM, later used by different companies and they made SQL available in there products according to there standards.

- SQL is a called Common Database Language, since it gets understand by every RDBMS
- SQL is a Command Based Language
- SQL is a Insensitive Language
- SQL is a Non Procedural Language

SQL Consist of following Sub Languages:

DATA QUERY OR RETRIVAL LANGUAGE {DQL OR DRL}

It is used to retrieve the data in different ways using "SELECT" Command.

DATA MANIPULATION LANGUAGE {DML}

Insertion of new rows, modifications to existing rows, removing unwanted rows, collectively known as DATA MANIPULATION LANGUAGE, It includes INSERT, UPDATE, DELETE and MERGE (introduced in 2008)

TRANSACTION CONTROL LANGUAGE {TCL}

This language supports to make a transaction permanent in a database or supports to cancel the transaction. It includes COMMIT, ROLLBACK, SAVE commands.

DATA DEFINATION LANGUAGE {DDL}

This language is used to CREATE, ALTER and DROP Database objects like tables, views, indexes, synonyms, store procedures, store functions, stored triggers. It includes CREATE, ALTER, TRUNCATE, DROP Commands.

DATA CONTROL LANGUAGE {DCL}

It is used to give rights on Database objects created by one user to get them access by other users it also supports to cancel the given rights, It includes GRANT, REVOKE, DENY Commands.

To display list of User Defined Tables

```
SELECT NAME FROM SYS.TABLES
SELECT NAME FROM SYS.SYSOBJECTS WHERE XTYPE = 'U'
```

To display list of System Tables

```
SELECT NAME FROM SYS.SYSOBJECTS WHERE XTYPE = 'S'
```

To display list of User Defined Views

```
SELECT NAME FROM SYS.SYSOBJECTS WHERE XTYPE = 'V'
```

To display STRUCTURE or DEFINITION or METADATA of a Table

```
SP_HELP < TABLE_NAME >
```

DEMO TABLES FOR ALL QUERIES IN THIS DOCUMENT

```
CREATE TABLE EMP ( EMPNO INT , ENAME VARCHAR(10), JOB VARCHAR(9),
MGR INT, HIREDATE DATETIME, SAL INT, COMM INT, DEPTNO INT )
```

```
INSERT INTO EMP VALUES(7369,'SMITH','CLERK',7902,'1980-12-17',800, NULL, 20)
INSERT INTO EMP VALUES(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30)
INSERT INTO EMP VALUES(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30)
INSERT INTO EMP VALUES(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20)
INSERT INTO EMP VALUES(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30)
INSERT INTO EMP VALUES(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30)
INSERT INTO EMP VALUES(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10)
INSERT INTO EMP VALUES(7788,'SCOTT','ANALYST',7566,'1982-12-09',3000,NULL,20)
INSERT INTO EMP VALUES(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10)
INSERT INTO EMP VALUES(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30)
INSERT INTO EMP VALUES(7876,'ADAMS','CLERK',7788,'1983-01-12',1100, NULL,20)
INSERT INTO EMP VALUES(7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30)
INSERT INTO EMP VALUES(7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20)
INSERT INTO EMP VALUES(7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10)

CREATE TABLE DEPT ( DEPTNO INT, DNAME VARCHAR(14), LOC VARCHAR(13) )

INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK')
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS')
INSERT INTO DEPT VALUES (30,'SALES','CHICAGO')
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON')

CREATE TABLE SALGRADE ( GRADE INT, LOSAL INT, HISAL INT )

INSERT INTO SALGRADE VALUES (1,  700, 1200)
INSERT INTO SALGRADE VALUES (2, 1201, 1400)
INSERT INTO SALGRADE VALUES (3, 1401, 2000)
INSERT INTO SALGRADE VALUES (4, 2001, 3000)
INSERT INTO SALGRADE VALUES (5, 3001, 9999)
```

DATA QUERY LANGUAGE

- This language supports to retrieve the data in 3 ways
  - SELECTION - retrieves the data of all columns.
  - PROJECTION - retrieves required columns.
  - JOINS - retrieves from multiple tables which always produce a new table.
- This language includes "SELECT" Command.

Syntax:

```
  SELECT * /LIST_OF_COLS[COL_ALIAS]/LITERALS/MATH EXPRESSIONS/FUNCTIONS
[   FROM TABLE1 [TABLE_ALIAS, TABLE2 TABLE_ALIAS...]]
[  WHERE CONDITIONS ]
[  GROUP BY COLUMN1 [COL2,COL3...] ] [ WITH ROLLUP / CUBE ]
[ HAVING CONDITIONS ]
[  ORDER BY COLUMN1 [COL2,COL3...] ] [ ASC / DESC ]
[  FOR XML MODES ]
```

Important points to be remembered:

- In a single SELECT statement data can be retrieved of 4096 columns from 255 tables.
- Clauses of SELECT statement should always be used as they are mentioned in the syntax.
- In SQL server selection & project can be done simultaneously.
- SELECT statement always gets executed from left to right.

Q) Display the entire department's information.

```
SELECT * FROM DEPT
```

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

Q) Display all the EMPLOYEES information.

```
SELECT * FROM EMP
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	800	NULL	20
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	1600	300	30
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	1250	500	30
4	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	1250	1400	30
6	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2850	NULL	30
7	7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.000	2450	NULL	10
8	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
9	7839	KING	PRESIDENT	NULL	1981-11-17 00:00:00.000	5000	NULL	10
10	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	1500	0	30
11	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
12	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	950	NULL	30
13	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
14	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

Q) Display all the SALGRADE information.

```
SELECT * FROM SALGRADE
```

	GRADE	LOSAL	HISAL
1	1	700	1200
2	2	1201	1400
3	3	1401	2000
4	4	2001	3000
5	5	3001	9999

Q) Display ENAME, JOB, SALARY, COMM and DEPTNO of all the EMPLOYEES.

```
SELECT ENAME,JOB, SAL, COMM, DEPTNO FROM EMP
```

Q) Display DEPTNO, DNAME of all the departments.

```
SELECT DEPTNO,DNAME FROM DEPT
SELECT * DNAME, LOC FROM DEPT
```

CLIENT/SERVER {Query Execution Process}

Step 1: Checks the syntax
Step 2: Compiles the query
Step 3: Generate execution plan

Step 4: Query would be executed

COLUMN ALIAS:

- It is the other name provided for a COLUMN.
- It is a temporary name provided for a COLUMN.
- It provides its effect only in the output of a query.
- It provides its usage till the query is in execution.
- It cannot be used in other clauses of SELECT statement except ORDER BY clause.
- When COLUMN ALIAS is specified directly it does not allow blank spaces and special characters except underscores.
- In order to have blank spaces and special characters it should be enclosed in **Double Quotes " " or Square Brackets [ ]**
- It can be specified in two ways:
  - With using "AS" Key word
  - With out using "AS" Key word

Examples:

```
SELECT EMPNO ECODE, ENAME EMPNAME FROM EMP
SELECT EMPNO AS ECODE, ENAME AS EMPNAME FROM EMP
SELECT EMPNO ECODE, SAL MONTHLY_SAL FROM EMP
SELECT EMPNO ECODE, SAL "MONTHLY PAY" FROM EMP
SELECT EMPNO ECODE, SAL [MONTHLY PAY] FROM EMP
```

	ECODE	MONTHLY PAY
1	7369	800
2	7499	1600
3	7521	1250
4	7566	2975
5	7654	1250

USING LITERALS IN A SELECT STATEMENT

- Literals refer to a constraint which means input provided for a query the same is provided in the output.
- A literal can be of numeric, string and date type data

Examples:

```
SELECT ENAME FROM EMP
SELECT 'ENAME' FROM EMP
SELECT 'ENAME' L1 FROM EMP
SELECT 'ENAME'
SELECT 'SQL' + 'SERVER' + '2005'
SELECT 'SQL'L1, 'SERVER'L2, '2005' L3
SELECT ENAME + ' WORKING AS ' + JOB RESULT FROM EMP
```

	RESULT
1	SMITH WORKING AS CLERK
2	ALLEN WORKING AS SALESMAN
3	WARD WORKING AS SALESMAN

In SQL Server "+" acts as a concatenation operator

USING MATHEMATICAL EXPRESSION IN SELECT STATEMENT

An expression which is consisting of **ARITHMETIC OPERATORS** like +, -, *, /, %

Examples:

```
SELECT 10 + 20 {FOR ADDITION}
SELECT 10 * 20 {FOR MULTIPLICATION}
SELECT 5/2 -- WILL TREAT AS INTEGER
SELECT 5/2.0
SELECT 10 + 5 * 6 M1, (10+5)* 6 M2
```

Q) Display ENAME, JOB, SAL and ANNUAL SALARY of all EMPLOYEES.

```
SELECT ENAME,JOB, SAL, SAL *12 ANNUALSAL FROM EMP
```

	ENAME	JOB	SAL	ANNUALSAL
1	SMITH	CLERK	800	9600
2	ALLEN	SALESMAN	1600	19200
3	WARD	SALESMAN	1250	15000
4	JONES	MANAGER	2975	35700
5	MARTIN	SALESMAN	1250	15000
6	BLAKE	MANAGER	2850	34200
7	CLARK	MANAGER	2450	29400

Q) Display ENAME, SAL, COMM and TOTAL SALARY of all EMPLOYEES.

```
SELECT ENAME, SAL, COMM, SAL + COMM TOTALSAL FROM EMP
```

	ENAME	SAL	COMM	TOTALSAL
1	SMITH	800	NULL	NULL
2	ALLEN	1600	300	1900
3	WARD	1250	500	1750
4	JONES	2975	NULL	NULL
5	MARTIN	1250	1400	2650
6	BLAKE	2850	NULL	NULL
7	CLARK	2450	NULL	NULL
8	SCOTT	3000	NULL	NULL
9	KING	5000	NULL	NULL
10	TURNER	1500	0	1500
11	ADAMS	1100	NULL	NULL
12	JAMES	950	NULL	NULL
13	FORD	3000	NULL	NULL
14	MILLER	1300	NULL	NULL

The above query calculates **TOTAL SALARY of those **EMPLOYEES** who are earning **COMMISSIONS** and all other **EMPLOYEES TOTAL SALARY** has been displayed as **NULL** values, since **VALUE + NULL =NULL**. It is applicable for all **ARITHMETIC OPERATORS.**

WHERE CLAUSE:

- This clause will restrict number of rows in the output of a query.
- This clause is used to compare the data by making condition in two ways. Filtering Conditions and Joining Conditions.
- At Filtering Condition data given by a user will be compared into a column.
- At Joining Condition data of one column will be compared with the data of another column.
- At Filtering Condition Comparison to string type data is not case sensitive and it should be placed in single query.
- Comparison to Numeric Type Data is direct or it can also be placed in single quotes.

OPERATORS:

RELATIONAL OPERATORS: >, <, >=, <=, {! = or <>} not equal to
LOGICAL OPERATORS: AND, OR, NOT
SPECIAL OPERATORS: IN, BETWEEN, IS, LIKE, DISTINCT, SAME/ANY, ALL, EXISTS
NOT + SPECIAL OPERATORS: NOTIN, NOT BETWEEN, ISNOT, NOT LIKE, NOT EXISTS

Q) Display all those EMPLOYEES who are working as **CLERKS.**

```
SELECT ENAME, JOB FROM EMP WHERE JOB = 'CLERK'
```

Q) Display all those EMPLOYEES who are working at Department Number 10

```
SELECT * FROM DEPT WHERE DEPTNO = 10
or
SELECT * FROM DEPT WHERE DEPTNO = '10'
```

Q) Display ENAME, JOB of those EMPLOYES who are not working as MANAGER

```
SELECT ENAME, JOB FROM EMP WHERE JOB != 'MANAGER'
```

Q) Display ENAME, JOB, and DEPTNO of those EMPLOYEES who are working as CLERK at DEPT NO 20.

```
SELECT * FROM EMP WHERE JOB = 'CLERK' AND DEPTNO = 20
```

Q) Display ENAME, JOB of those EMPLOYEES who are working as CLERK, ANALYST.

```
SELECT ENAME, JOB FROM EMP WHERE JOB = 'CLERK' OR JOB = 'ANALYST'

SELECT ENAME, JOB FROM EMP WHERE JOB IN ('CLERK','ANALYST')
```

Q) Display EMPNO, ENAME, DEPTNO of those EMPLOYEES who are working at 10, 30 Departments.

```
SELECT EMPNO, ENAME, DEPTNO FROM EMP WHERE DEPT IN (10,30)
```

IN:

- This operator compares list of similar type values.
- It works like "**OR**" Logical operator.
- It can be used for any number of times.

- The list should contain only one column values; multiple columns data is invalid.

Q) Display ENAME, SAL of EMPLOYEE SAL Greater than or equal to 2000 & SAL less than or equal to 3000.

```
SELECT ENAME, SAL FROM EMP WHERE SAL >= 2000 AND SAL <=3000

SELECT ENAME, SAL FROM EMP WHERE SAL BETWEEN 2000 AND 3000
```

BETWEEN:

- It is used to compare range of values
- Compares range of values including the specified data
- It can also be used for any number of times

Q) Display ENAME, JOB, DEPTNO of those Employees who are not working as CLERK, ANALYST.

```
SELECT ENAME, JOB, DEPTNO FROM EMP WHERE JOB NOT IN ('CLERK','ANALYST')
```

Q) Display all those Employees whose salary is not in a range of 1000 and 3000.

```
SELECT * FROM EMP WHERE SAL NOT BETWEEN 1000 AND 3000
```

Q) Display all the Employees who are working as CLERK, ANALYST and SAL is greater than 1000.

```
SELECT * FROM EMP WHERE JOB = 'CLERK' OR 'ANALYST' AND SAL > 1000

SELECT * FROM EMP WHERE JOB IN ('CLERK','ANALYST') AND SAL > 1000
```

Q) Display ENAME, SAL, ANNUAL SAL of those Employees whose annual salary is in range of 18000 and 36000.

```
SELECT ENAME, SAL, SAL*12 AS ANNUALSAL FROM EMP
 WHERE SAL*12 BETWEEN 18000 AND 36000
```

Q) Display ENAME, JOB, SAL, COMM and DEPTNO who are working as SALESMAN at DEPTNO 30 and there commission is > half of their salary.

```
SELECT * FROM EMP WHERE JOB = 'SALESMAN' AND DEPTNO = 30 AND COMM > SAL/2
```

ANSI NULLS:

- Working with **IS** Operator.
- This operator is exclusively used for comparing NULL values.
- It can be used any number of times.
- SQL server also supports to compare the NULL values using relation operator.
- Following SET Command should be used

  `SET ANSI_NULLS OFF | ON` [Default it is ON]
- It should be turned to off when NULL value is compared with relational operator.

Q) Display all those Employees who are not earning commission.

```
SELECT * FROM EMP WHERE COMM IS NULL
```

Q) Display all those Employees whose are earning commission.

```
SELECT * FROM EMP WHERE COMM IS NOT NULL

SELECT * FROM EMP WHERE NOT COMM IS NULL
```

Q) Display EMPNO, ENAME & MGR of those Employees who don't have manager to report.

```
SELECT * FROM EMP WHERE MGR IS NULL
```

Q) Display EMPNO, ENAME & MGR of those Employees who have managers to report.

```
SELECT * FROM EMP WHERE MGR IS NOT NULL
```

Q) Display EMPNO, ENAME, MGR, SAL and COMM of those Employees who have managers to report and do not earn commission.

```
SELECT * FROM EMP WHERE MGR IS NOT NULL AND COMM IS NULL

SET ANSL_NULLS OFF
SELECT * FROM EMP WHERE COMM != NULL
SELECT * FROM EMP WHERE COMM = NULL
SELECT * FROM EMP WHERE MGR = NULL
SELECT * FROM EMP WHERE NULL = NULL
```

WILD CHARACTERS:

These characters are used to provide pattern matching when data is searched into a column SQL SERVER supports to work with following characters.

- "_" – it is used for single characters
- "%" - it is used for group of characters
- "-"– it is used for range of characters

LIKE OPERATOR:

- It is used for comparing the data which is enclosed with wild characters. It can be used for any number of times.
- This operator when compares the data using wild characters it should be enclosed in single quotes.
- Comparison to any type data should be placed in single quotes
- If data in a column is enclosed with wild characters to search the data escape option should be specified

```
SELECT * FROM EMP WHERE ENAME LIKE 'J%'
SELECT * FROM EMP WHERE ENAME LIKE '%S'
SELECT * FROM EMP WHERE ENAME LIKE 'J%S'
SELECT * FROM EMP WHERE ENAME LIKE '%A%'
SELECT * FROM EMP WHERE ENAME LIKE '%A%E%'
SELECT * FROM EMP WHERE ENAME LIKE 'J[_]%'
SELECT * FROM EMP WHERE ENAME LIKE 'J\_%' ESCAPE '\'
```

****In ESCAPE we can use any characters except WILD CHARACTERS.**

```
SELECT * FROM EMP WHERE ENAME LIKE '[A-F]%'
SELECT * FROM EMP WHERE ENAME NOT LIKE '[A-F]%'
SELECT * FROM EMP WHERE ENAME LIKE '[^A-F]%'
SELECT * FROM EMP WHERE ENAME LIKE '___'
SELECT * FROM EMP WHERE ENAME LIKE '%?_%' ESCAPE '?'
```

DISTINCT OPERATOR:

- Eliminates duplicates.
- Always arranges the data in ASCENDING ORDER.
- It should be used immediately after SELECT command.
- It includes NULL VALUES.
- When multiple columns are used with DISTINCT operator it eliminates only if all the columns contains duplicates.

```
SELECT DISTINCT DEPTNO FROM EMP
SELECT DISTINCT JOB FROM EMP
SELECT DISTINCT COMM FROM EMP
SELECT DISTINCT MGR FROM EMP WHERE MGR IS NOT NULL
SELECT DISTINCT DEPTNO, JOB FROM EMP
```

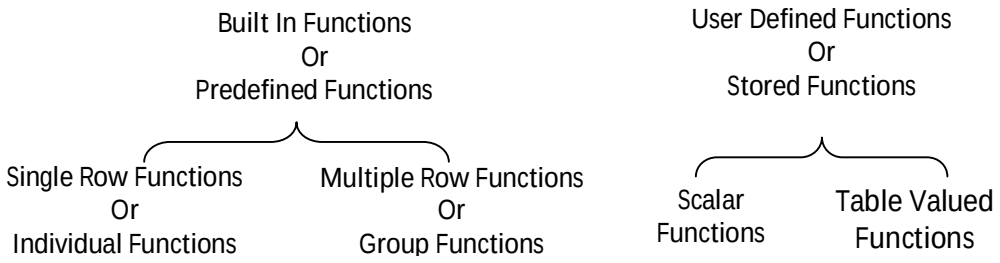
ORDER BY:

- This clause is used to arrange the data in ASCENDING / DESCENDING order.
- By default it arranges the data in ASCENDING order.
- Order by clause will work after the execution of "SELECT" statement.
- Order by clause will arrange the data in order at buffer location, hence it is temporary.
- It can arrange the data in order based on COLUMN_NAME or COLUMN_ALIAS_NAME or COLUMN_POSITION {according to query} to specify order type.
  ASC ----- ASCENDING & DESC ---- DESCENDING
- When multiple columns are used at **ORDER BY** clause. First column will be fully arranged in order and other columns will be arranged based on previous column.
- When multiple columns are used at **ORDER BY** clause, they can be specified with same order type or different order type.

```
SELECT * FROM EMP ORDER BY SAL
SELECT EMPNO, ENAME, JOB, SAL AS PAY FROM EMP ORDER BY PAY
SELECT EMPNO, ENAME, JOB, SAL AS PAY FROM EMP ORDER BY 3
SELECT EMPNO, ENAME, JOB, SAL AS PAY FROM EMP ORDER BY 3,4 DESC
SELECT EMPNO, ENAME, JOB, SAL AS PAY FROM EMP ORDER BY 3 ASC, 4 DESC
```

FUNCTIONS:

It is a predefined program which returns one value, returned value can be of numeric, string data, data type functions can be used for calculations



SINGLE ROW FUNCTION: This function will process one row at a time and returns one value.

- Mathematical Functions
- String Functions
- Date & Time Functions
- Data Conversion {Type Casting Functions}
- Ranking Functions {Windows Functions}
- Meta Data Functions
- Other Functions

MULTIPLE ROW FUNCTION: This function will process multiple rows at a time and returns one value.

- Aggregate Functions

SQL SERVER supports to work with 12 categories of Functions and they are found at following paths

- Click on Object Explorer – Data Bases
- Select the Data Base - TESTDB
- Check under Programmability Functions.
- Check under System Functions.

MATHEMATICAL FUNCTIONS:

This function will take input as numbers and will return output as numbers

`ABS (NUMBER)` - returns unsigned value of a given number (Number - Argument or Parameter.

```
SELECT ABS(-890), ABS(17)
```

`SQRT (NUMBER)` - Returns square root of a given positive number.
`SQUARE (NUMBER)` - Returns square of a given value
`POWER ( NUMBER(Base), NUMBER(Exponent))` - It will find the power of a given number.

```
SELECT POWER (2,5)
```

`SIGN ( NUMBER )`

Returns 1 if a number is positive
Returns -1 if a number is negative
Returns 0 if a number is zero

`PI()` - Returns PI value

`SIN (NUMBER)` - By default this function will take input given in radians, hence radians should be converted to degrees by a standard formula is PI ()/180

```
SELECT SIN (30* PI()/180)
```

****Functions used with in other function refers to FUNCTION CASCADING**

```
ROUND ( NUMBER, NUMBER , [NUMBER])
```
Argument1, Argument2, Argument3 (optional for TRUNCATE)
By default Argument3 is ZERO

```
ROUND(1234.5678,2,1) --- 1234.5600
ROUND(1234.5678,2) ----- 1234.5700
ROUND(1234.5638,2) ----- 1234.5600
ROUND(1234.5678,1) ----- 1234.6000
ROUND(1234.5678,0) ----- 1235.0000
ROUND(1234.2678,0) ----- 1234.0000
ROUND(1234.5678,-2)----- 1200.0000
ROUND(5678.123,-2) ----- 5700.0000
ROUND(1234.5678,2,1) --- 1234.5600
ROUND(1234.5678,1,1) --- 1234.5000
ROUND(1234.5678,0,1) --- 1234.0000
ROUND(5/2.0,0) --- 3.00
ROUND(5/2.0,0,1) --- 2
```

This function is used to round a given number with respect to Integer value or Decimal value based on user required number of digits. If 3rd argument is specified other than zero it works as **TRUNCATE**.

`CEILING(NUMBER)` – This function will increment a given number to its nearest integer. Based on any digit in decimal points is greater than zero.

```
CEILING (123.000) ---- 123
CEILING (123.010) ---- 124
CEILING (123.456) ---- 124
```

```
FLOOR (NUMBER) – It decreases a nearest integer
FLOOR (-123.456) ---- {-124}
```

STRING FUNCTIONS: This are used to perform different operations on STRING DATA TYPE.

`LEN(TEXT)` - Returns number of characters.

```
SELECT ENAME, LEN(ENAME) FROM EMP WHERE LEN(ENAME)  > 5
```

```
LOWER(STRING) – Converts characters to Lower case.
UPPER(STRING) – Converts characters to Upper case.
```

```
ASCII(STRING) – returns ASCII code of the given character
      ASCII('A')----65
      ASCII('P')----80
```

```
UNICODE( STRING ) – Returns UNICODE of a given string.
```

```
CHAR ( NUMBER ) – Returns character of a given ASCII code
CHAR (65) -- A   { Only 0-255 characters are available}
```

```
NCHAR(NUMBER)
NCHAR(256) {1-65535 Characters are available}
```

```
LEFT(STRING,NUMBER) – Extracts number of characters from left side of a string
LEFT('COMPUTER',3) ---- COM

RIGHT( STRING, NUMBER) – Extracts number of characters from right side of a string
RIGHT('COMPUTER',3)-----TER

SUBSTRING (STRING, STARTPOSITION, NUMBER OF CHAR'S) – It extracts required string from a given string based on start position and number of characters.

SUBSTRING ('COMPUTER',4,3)–PUT

LTRIM(STRING) - Removes blanks spaces from left side of a string.
RTRIM(STRING)- Removes blank spaces from right side of a string.

REPLICATE(STRING, NUMBER) - Displays a given string for n times.
REPLICATE('COMM',2)

REVERSE(STRING) - It will reverse a given string.
SELECT REVERSE('COMM')--- MMOC
REPLACE (STRING1, STRING2, STRING3)
String1 ---- given string, String2 ---- search string, String3 ---- replace string

REPLACE ('COMPUTER','UT','IR')
It will search for String2 in String1 and replaces with String3 at all occurrences of String 2 in String1

STUFF(STRING1, NUMBER1, NUMBER2, STRING2)
String1 --- Given String, Number 1--- Start Position, Number2 --- Number of Characters
String2 --- Replace

STUFF('COMPUTER',5,2,'IL') -- COMPILER

CHARINDEX(STRING1, STRING2, [NUMBER])
String1 --- Search String, String2 --- Given String, Number --- Position

It will search for string1 in string2 and returns position of string1, if it exists else returns zero, by default it will search from first character. It also supports to search for character/string after a specific position

SELECT CHARINDEX('R','COMPUTER',7)

SELECT REPLICATE ('*', GRADE) FROM SALGRADE

Q) Display all the employees from the EMP table with half of the string in upper case and next string in lower case
SELECT LEFT (ENAME, CEILING (LEN (ENAME)/2.0)) +
 LOWER(RIGHT (ENAME, CEILING (LEN (ENAME)/2))) FROM EMP

Q) Display all Employees whose name contains 3rd Character as 'L'.

SELECT ENAME FROM EMP WHERE SUBSTRING (ENAME, 3,1)= 'L'

SELECT ENAME FROM EMP WHERE CHARINDEX ('L', ENAME, 3) = 3
```

Q) Display all Employees whose name contains A at second occurance.

```
SELECT ENAME FROM EMP
 WHERE CHARINDEX ('A', ENAME, CHARINDEX ('A', ENAME) +1 !=0
```

DATE & TIME FUNCTIONS:

These functions are used to perform operations on date, time data type. When date type data is retrieved it will be in the following format.

```
CENTURY: YEAR: MONTH: DATE: HOURS: MINUTES: SECONDS
```

`GETDATE()`: It returns current system DATE & TIME of the Server.

```
SELECT GETDATE()
```

ARITHMETIC OPERATIONS on data & time data type "+", "-" are the operations which can be used for performing operations on date type data.

DATE + NUMBER = DATE
DATE – NUMBER = DATE
DATE – DATE = NUMBER [Number of days]

```
SELECT HIREDATE, HIREDATE+15, HIREDATE-28 FROM EMP
```

DAY (DATE) -------- extracts days from date
MONTH (DATE) ----- extracts months from date
YEAR (DATE) ------ extracts year from date

```
SELECT DAY(GETDATE())
SELECT MONTH(GETDATE())
SELECT YEAR(GETDATE())
```

`DATEADD(DATEPART , NUMBER , DATE)` - It is used to add number of days, months, years to a given date.

DATEPART can be specified with this format - DD: MM: YY: HH: MI: SS: DW

```
SELECT DATEADD(DAY,10,GETDATE())
SELECT DATEADD(DD,10,GETDATE())
SELECT DATEADD(MONTH,10,GETDATE())
SELECT DATEADD(MM,10,GETDATE())
SELECT DATEADD(YEARS,10,GETDATE())
SELECT DATEADD(YY,10,GETDATE())
```

`DATEDIFF( DATEPART, STARTDATE, ENDDATE)` - It returns difference between 2 dates in terms of days, months & years.

```
SELECT ENAME, DATEDIFF(DD,HIREDATE,GETDATE()) DAYS FROM EMP
```

`DATEPART(DATEPART,DATE)`- It extracts individual parts of a table.
`DATEPART(DD,GETDATE())` – MM: YY: HH: MI: SS: DW
`DATENAME(DATEPART,DATE)` - In this function month name, day name will be extracted other date parts providing same output.

```
DATENAME(MM,GETDATE()) ---- JUNE(Based on the current server date)
DATENAME(DW,GETDATE()) ---- MON
```

Q) Display all those Employees data that have been hired on Sunday.

```
SELECT ENAME, HIREDATE FROM EMP WHERE DATENAME(DW,HIREDATE) = 'SUNDAY'
```

Q) Display all those Employees who have been hired in the month of December.

```
SELECT ENAME,HIREDATE FROM EMP WHERE DATENAME(MM,HIREDATE)= 'DECEMBER'
```

Q) Display all those Employees who have been hired in second half of the week.

```
SELECT ENAME, HIREDATE FROM EMP
 WHERE DATEPART (DW, HIREDATE) > = ROUND(7/2.0,0)
```

DATA CONVERSION:

- This function will support to convert the data stored with one type to the other type.
- SQL SERVER will convert the data with the support of following two functions.

```
CAST(SOURCE_DATA AS TARGET_DATA_TYPE)
CONVERT(TARGET_DATA_TYPE, SOURCE_DATA [NUMBER])
```

```
SELECT ENAME + ' WORKING AS ' + JOB + ' IN DEPT ' + CAST(DEPTNO AS
VARCHAR(3)) FROM EMP
```

**** String and String can only be concatenated.**
**** String and Integer cannot be concatenated.**

```
SELECT 5/ CAST(2 AS FLOAT)
SELECT 5/ CONVERT(FLOAT,2)
SELECT CONVERT(VARCHAR(30), GETDATE(),0)
SELECT CONVERT(VARCHAR(30), GETDATE(),8) for time
```

**** 0 stands for Date formats, there are more than 100 Date formats.**
**** 8 & 108 are particularly assigned for time format**

RANKING OR WINDOW FUNCTIONS

- These functions have been introduced from SQL 2005.
- This are used to calculate ranks on any type of data.
- These functions require a support of OVER clause, which requires a specification of ORDER BY clause, followed by ORDER type.
- This function cannot be used at WHERE clause.

SQL SERVER will support to work with following Rank functions

`ROW_NUMBER()` – Provides unique row number for each row
`RANK()` – It calculates ranks with gaps
`DENSE_RANK()` – It calculates ranks with out gaps
`NTILE(INT)` – Provides the data in to groups / blocks. It will group the data based on the user specified number and number of rows available in a table.

```
SELECT EMPNO, ENAME, SAL, ROW_NUMBER() OVER ( ORDER BY EMPNO ASC) RN
  FROM EMP

SELECT EMPNO, ENAME, SAL, RANK() OVER(ORDER BY SAL DESC) RNK FROM EMP

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

SELECT EMPNO, ENAME, SAL, NTILE(4) OVER( ORDER BY EMPNO) NT FROM EMP
```
****The data will be grouped into four groups based on given NTILE number.**

OTHER FUNCTIONS

```
ISNULL(EXPRESSION1, EXPRESSION2)
```

This Function will search for NULL values in Expression1 and Substitutes Expression2 at all the rows of Expression1 where NULL values are found, Expression2 is depended on data type of Expression1.

```
SELECT ENAME, SAL, COMM, ISNULL(COMM,0) RS FROM EMP

SELECT ENAME, SAL, COMM, ISNULL(ENAME,'UNKNOWN') UN FROM EMP

SELECT ENAME ,SAL, COMM, ISNULL ( CAST(COMM AS VARCHAR(4),'NC') RS
  FROM EMP

SELECT ENAME,SAL,COMM,SAL+ISNULL(COMM,0) RS FROM EMP
```

`COALESCE(EXPRESSION1,EXPRESSION2……EXPRESSIONN)` - It will search for NULL values and Substitutes Expression2 at all rows of Expression1 where NULL values are found. It Substitutes Expression3 if Expression2 contains NULL values so on it can have any number of expressions.

```
SELECT EMPNO,ENAME,MGR,SAL,COMM COALESCE(COMM,SAL,MGR,EMPNO,25) RS
  FROM EMP
```

`NULLIF( EXPRESSION1, EXPRESSION2)` - It is used to compare two expressions of any data type. If equal it returns NULL, if not equal returns value of Expression1

```
SELECT NULLIF(10,10) -----NULL
SELECT NULLIF(10,90) -----10
SELECT NULLIF('X','Y')--- X
SELECT ENAME,LEN(ENAME)L1,JOB,LEN(JOB)L2 NULLIF(LEN(ENAME),LEN(JOB))RES
  FROM EMP
```

CASE EXPRESSION

It supports to specify multiple conditions provided with their respective results. If no condition is satisfied then it will provide default result.

Syntax:

```
CASE EXPRESSION
     WHEN CONDITION1 THEN RESULT1
   [ WHEN CONDITION2 THEN RESULT2 ]
ELSE
     DEFAULT_RESULT
END [ALIAS_NAME]
```

Q) Display ENAME, SAL and INCREMENT the salary with difference %'s based on category of JOB.
Manager – 18%, Clerk – 15%, Others – 12%

```
SELECT ENAME,SAL,JOB,
  CASE JOB
    WHEN 'MANAGER' THEN SAL*18/100
    WHEN 'CLERK' THEN SAL * 15/100
  ELSE
    SAL*12/100
  END INCR
FROM EMP
```

Q) Display ENAME, DEPTNO, SAL and INCREMENT the Salary with different %'s on category of Dept No 10 – 20%, Dept no 20 – 18%, Dept no 30 – 15%

```
SELECT ENAME, DEPTNO, SAL,
  CASE DEPTNO
    WHEN 10 THEN SAL* 20/100
    WHEN 20 THEN SAL * 18/100
    WHEN 30 THEN SAL*15/100
  END INCR
FROM EMP
```

MULTIROW FUNCTIONS OR GROUP FUNCTIONS

AGGREGATE FUNCTIONS:

- This function will process multiple rows at a time and returns one value.
- SQL SERVER supports to work with following functions
- SUM (EXPRESSION) - finds the sum of values in given expression.
- AVG (EXPRESSION) - first finds the sum and then divide with number of values in the expression.
- MAX (EXPRESSION) - finds the maximum value in the given expression.
- MIN (EXPRESSION) - finds the minimum value in the given expression.
- COUNT (EXPRESSION) --- returns number of values in a expression including duplicates.
- COUNT (DISTINCT (EXPRESSION) - returns number of values in an expression excluding duplicates.
- COIUNT (*) - returns number of rows.
- COUNT_BIG (EXPRESSION) - returns number of values.
- GROUPING (EXPRESSION) - returns zero or one.
    - Returns ZERO if there is no group
    - Returns ONE if there is a group data

POINTS TO BE REMEMBERED

- Aggregate Function will exclude NULL values.
- When a SELECT statement is enclosed with Aggregate Functions, along with them writing expressions that returns multiple rows is invalid.
- Aggregate functions cannot be used in where clause.
- In SQL SERVER Aggregate Functions cannot be cascaded.

```
SELECT SUM(SAL) FROM EMP
SELECT AVG(SAL) FROM EMP
```

```
SELECT SUM(COMM) R1, AVG(COMM) R2, COUNT(COMM)R3 FROM EMP
SELECT SUM(SAL), MAX(SAL) FROM EMP WHERE DEPTNO = 30
SELECT COUNT(SAL) FROM EMP
SELECT COUNT(DISTINCT(SAL) FROM EMP
SELECT COUNT(*) FROM EMP
```

To Count the Records

```
SP_SPACEUSED EMP
SELECT @@ROWCOUNT

SELECT SUM(SAL) , COUNT(*) FROM EMP WHERE JOB = 'SALESMAN'
SELECT DEPTNO,SUM(SAL),COUNT(*) FROM EMP GROUP BY DEPTNO

SELECT DEPTNO, JOB,SUM(SAL), COUNT(*) FROM EMP GROUP BY JOB, DEPTNO
```

GROUP BY: This clause performs two tasks

- Eliminates duplicates and always arranges the data in ascending order.
- The column used in group by clause, the same column can be used for displaying the output.
- When multiple columns are used it will eliminate if all column contains duplicates.

```
SELECT SUM(SAL), COUNT(*) FROM EMP GROUP BY DEPTNO
SELECT DEPTNO, SUM(SAL), COUNT(*) FROM EMP GROUP BY DEPTNO
SELECT JOB, SUM(SAL), COUNT(*) FROM EMP GROUP BY JOB
SELECT MGR, COUNT(*) FROM EMP GROUP BY MGR
SELECT DEPTNO, JOB, SUM(SAL), COUNT(*) FROM EMP GROUP BY JOB , DEPTNO
```

Q) Display DEPTNO, total salary and number of employee's related to Dept No 20.

```
SELECT DEPTNO, SUM(SAL), COUNT(*) FROM EMP
 WHERE DEPTNO = 20 GROUP BY DEPTNO
```

ROLLUP AND CUBE OPERATORS

ROLLUP:

- This operator should be used only at GROUP BY clause.
- It will calculate individual totals, group totals, and additional row is generated for grand totals.

```
SELECT DEPTNO, SUM(SAL), COUNT(*) FROM EMP GROUP BY DEPTNO WITH ROLLUP

SELECT DEPTNO, JOB, SUM(SAL), COUNT(*) FROM EMP
 GROUP BY DEPTNO, JOB WITH ROLLUP

SELECT DEPTNO, JOB, SUM(SAL),COUNT(*) , GROUPING(JOB) FROM EMP
 GROUP BY DEPTNO, JOB WITH ROLLUP

SELECT DEPTNO ,
  CASE GROUPING(JOB)
    WHEN 0 THEN JOB
    WHEN 1 THEN "ALL JOBS"  -- Replaces NULLS With The Value
  END DEP, SUM(SAL), COUNT(*)
FROM EMP GROUP BY DEPTNO, JOB WITH ROLLUP
```

CUBE:

- This operator is also used only at GROUP BY clause.
- It will support to perform individual totals, group totals, regroup totals and additional row is generated for grand totals.

```
SELECT DEPTNO, JOB, SUM(SAL), COUNT(*) FROM EMP
 GROUP BY DEPTNO, JOB WITH CUBE
```

Q) Display total salaries of individual departments in a single row.

```
SELECT
   SUM (CASE DEPTNO WHEN 10 THEN SAL END) "D10",
   SUM (CASE DEPTNO WHEN 20 THEN SAL END) "D20",
   SUM (CASE DEPTNO WHEN 30 THEN SAL END) "D30"
  FROM EMP

SELECT DISTINCT JOB,
   SUM (CASE DEPTNO WHEN 10 THEN SAL END) "D10",
   SUM (CASE DEPTNO WHEN 20 THEN SAL END) "D20",
   SUM (CASE DEPTNO WHEN 30 THEN SAL END) "D30",
   SUM (SAL) "TOTAL SAL"
  FROM EMP GROUP BY JOB
```

****This type of queries is called as CROSS TAB QUERIES or MATRIX FORMAT QUERIES.**

HAVING CLAUSE:

- It is used to make conditions on grouped data.
- It also supports to make conditions on columns those columns which are used at group by clause.

Q) Display Dept No, Total Salary of those departments where total salary is more than 9000.

```
SELECT DEPTNO,SUM(SAL) FROM EMP GROUP BY DEPTNO HAVING SUM(SAL) > 9000
```

Q) Display Dept No, total salary of those departments where total salary is more than 9000 and belong to dept no 20.

```
SELECT DEPTNO,SUM(SAL) FROM EMP GROUP BY DEPTNO
HAVING DEPTNO = 20  AND SUM(SAL) > 9000
```

Q) Display the departments where more than 3 Employees are working.

```
SELECT DEPTNO, COUNT(*) FROM EMP
GROUP BY DEPTNO HAVING COUNT(*) >= 3
```

SET JOINS OR SET OPERATORS:

- It is used to Join Rows that are returned by two or more queries.
- Queries can be on same or different tables.
- Columns in queries can be same or different.
- When columns are different, data types should be same.

- When columns are different in output, it will provide that column name which is in first query.
- To arrange the data in ascending order or in descending order order by clause should be associated to last query.
- When columns are different in queries column of first query should be used at order by clause.

SQL SERVER supports following **SET** Operators.

UNION ALL

- Does not eliminate duplicates.
- Does not arrange the data in Ascending Order.

```
SELECT JOB FROM EMP WHERE DEPTNO = 10
 UNION ALL
SELECT JOB FROM EMP WHERE DEPTNO = 20

SELECT JOB FROM EMP WHERE DEPTNO = 10
 UNION ALL
SELECT JOB FROM EMP WHERE DEPTNO = 20 ORDER BY JOB

SELECT JOB FROM EMP WHERE DEPTNO = 10
 UNION ALL
SELECT DNAME FROM DEPT ORDER BY JOB
```

UNION

- Eliminates duplicates.
- It always arranges the data in ascending order.
- When multiple columns are used, it will eliminate only if all the columns contain duplicates.

```
SELECT JOB FROM EMP WHERE DEPTNO = 10
 UNION
SELECT JOB FROM EMP WHERE DEPTNO = 20

SELECT JOB FROM EMP WHERE DEPTNO = 10
 UNION
SELECT JOB FROM EMP WHERE DEPTNO = 20 ORDER BY JOB
SELECT JOB FROM EMP WHERE DEPTNO = 10
 UNION
SELECT DNAME FROM DEPT ORDER BY JOB

SELECT JOB, SAL FROM EMP WHERE DEPTNO =10
 UNION
SELECT JOB,SAL FROM EMP WHERE DEPTNO = 20
```

INTERSECT

- Extracts common rows from two or more result sets.
- Eliminates duplicates.
- Always arranges in ascending order.

```
SELECT JOB FROM EMP WHERE DEPT NO=10
INTERSECT
SELECT JOB FROM EMP WHERE DEPT NO=20
```

EXCEPT

- It always produces output from first query, except the rows that are found at the result set of 2nd query.
- It will eliminate duplicates.

```
SELECT JOB FROM EMP WHERE DEPT NO=10
EXCEPT
SELECT JOB FROM EMP WHERE DEPT NO=20

SELECT JOB FROM EMP WHERE DEPT NO=20
EXCEPT
SELECT JOB FROM EMP WHERE DEPT NO=10

SELECT JOB FROM EMP WHERE DEPTNO = 10
SELECT JOB FROM EMP WHERE DEPTNO = 20
SELECT JOB FROM EMP WHERE DEPTNO = 30
SELECT DNAME FROM DEPT

SELECT JOB FROM EMP WHERE DEPT NO=10
EXCEPT
SELECT JOB FROM EMP WHERE DEPT NO=10
UNION
SELECT JOB FROM EMP WHERE DEPT NO=10
```

JOINS

- It is a process of joining columns from 2 or more tables, which always produces a new table
- Joining conditions is must
- To join n tables n-1 join conditions are required
- A joining condition is followed by any number of filtering conditions.

Non ANSI joins (These are implemented through operators)

- Inner Join (Equi join, Non Equi Join)
- Outer Join (Left Outer Join, Right Outer Join, Cartesian Join, Self Join)

ANSI Joins (These are implemented through key words)

- Inner Join or Join, Cross join
- Outer Joins (Left Outer Join, Right Outer Join, Full Outer Join)

NON ANSI JOINS - INNER JOIN

EQUI JOIN:

- This Join concept supports to join the columns from two or more tables by making a joining condition using “equal to” operator.
- To work with equi join tables should contain at least one common column with respect to data and data type.
- If tables are containing common column names, to use them in a query they should be preceded by table name or table alias name.

TABLE ALIAS:

- Table Alias can be used under any clause.
- It is other name provided for a table.
- It can be specified with or without as keyword.

- It is a temporary name provided for a table, which provides its usage till the query is in execution.
- It is flexible in usage.
- It can be used anywhere in SELECT statement.

```
SELECT EMPNO, ENAME, JOB, SAL, E.DEPTNO, DNAME, LOC
  FROM EMP, DEPT WHERE DEPTNO = DEPTNO

SELECT EMPNO, ENAME, JOB, SAL, E.DEPTNO, DNAME, LOC
  FROM EMP E, DEPT D WHERE E.DEPTNO = D.DEPTNO

SELECT EMPNO, ENAME, JOB, SAL, E.DEPTNO, DNAME, LOC
  FROM EMP AS E, DEPT AS D WHERE E.DEPTNO = D.DEPTNO
```

To connect two or more Databases and Tables

```
SELECT TESTDB.DBO.EMP.EMPNO, TESTDB.DBO.EMP.ENAME, TESTDB.DBO.EMP.JOB,
       TESTDB.DBO.EMP.SAL, TESTDB.DBO.DEPT.DEPTNO,
       TESTDB2.DBO.DEPT.DNAME, TESTDB2.DBO.DEPT.LOC
  FROM TESTDB.DBO.EMP, TESTDB2.DBO.DEPT
 WHERE TESTDB.DBO.EMP.DEPTNO = TESTDB2.DBO.DEPT.DEPTNO
```

NON EQUI JOIN: At these joining concept columns from two or more tables can be joined by making a joining condition other than equal to operator.

```
SELECT E.*, GRADE FROM EMP E, SALGRADE S
 WHERE SAL BETWEEN LOSAL AND HISAL

SELECT E.*, GRADE FROM EMP E, SALGRADE S
 WHERE SAL >= LOSAL AND SAL <= HISAL
```

Q) Display EMPNO, ENAME, DNAME, LOC, and GRADE of all the Employees.

```
SELECT EMPNO, ENAME,D.DNAME,D.LOC,S.GRADE
  FROM EMP E , DEPT D , SALGRADE S
 WHERE E.DEPTNO = D.DEPTNO
   AND E.SAL BETWEEN S.LOSAL AND S.HISAL
```

	EMPNO	ENAME	DNAME	LOC	GRADE
1	7369	SMITH	RESEARCH	DALLAS	1
2	7499	ALLEN	SALES	CHICAGO	3
3	7521	WARD	SALES	CHICAGO	2
4	7566	JONES	RESEARCH	DALLAS	4
5	7654	MARTIN	SALES	CHICAGO	2
6	7698	BLAKE	SALES	CHICAGO	4
7	7782	CLARK	ACCOUNTING	NEW YORK	4
8	7788	SCOTT	RESEARCH	DALLAS	4
9	7839	KING	ACCOUNTING	NEW YORK	5
10	7844	TURNER	SALES	CHICAGO	3

Q) Display ENAME, JOB, SAL, COMM, DEPTNO, DNAME, LOC & GRADE of all those Employees who are working at "ACCOUNTING, SALES" and they have managers to report and they don't earn commission and their Salary more than 1000, arrange the data in ascending order.

```
SELECT ENAME,JOB,SAL,COMM,MGR,E.DEPTNO,D.DNAME,D.LOC,S.GRADE
  FROM EMP E, DEPT D, SALGRADE S
 WHERE E.DEPTNO = D.DEPTNO
   AND E.SAL BETWEEN S.LOSAL AND S.HISAL
   AND D.DNAME IN('ACCOUNTING','SALES')
   AND E.MGR IS NOT NULL
   AND E.COMM IS NULL
   AND E.SAL > 1000 ORDER BY E.DEPTNO
```

SELF JOIN:

- At this joining concept single table will be used for more than once differentating them with the Alias Name.
- Data is retrived and joining condition is made in a single table.

Q) Display EMPNO, ENAME, MGR, MGRNAME for those Employees who have managers to report.

```
SELECT M.EMPNO,M.ENAME,M.MGR,E.ENAME FROM EMP E , EMP M
 WHERE E.EMPNO = M.MGR
```

CARTESIAN JOINS:

- At this joining concept, Columns from two or more tables can be joined without using any joining condition.
- It provides mutliplication of Rows among the tables used for joins

```
SELECT EMPNO, ENAME, JOB, E.DEPTNO, D.DNAME FROM EMP E,  DEPT D
```

OUTER JOINS :

- At this concept of joining, Columns from two or more tables with respect to rows can be joined that gets matched and unmatched records with the joining condition.
- "*" Operator is used for Outer Joins.
- SQL Server 2005 is restriced with the implementation of Outer joins through operator
- In order to implement outer joins in SQL Server 2005 data base compatability level should be used by a predefined procedure called SP_DBCMPTLEVEL.

Syntax :

```
SP_DBCMPTLEVEL @DBNAME = <'DATABASE_NAME'>
               @NEW_CMPTLEVEL = '80'
```

For SQL SERVER Version 7.00 = 70 should be used
For SQL SERVER Version 2000 = 80 should be used
For SQL SERVER Version 2005 = 90 should be used

SQL Server supports two types of Outer Joins through NON ANSI JOINS

LEFT OUTER JOIN: At this joining concept, Outer Join operator is placed on left side of joining condition, it will extract matched and unmatched rows from the table which is placed on LEFT side of Joining condition and substitutes NULL Values for the columns of a table which is on RIGHT side of the Joining Condition.

Syntax:

```
      SELECT * FROM TABLE1 , TABLE2
       WHERE TABLE1.COLUMN1 *= TABLE2.COLUMN2

SELECT EMPNO, ENAME, JOB, SAL, D.DNAME
  FROM EMP E , DEP D WHERE E.DEPTNO *= D.DEPTNO

SELECT EMPNO, ENAME, JOB, SAL, D.DNAME
  FROM EMP E , DEP D WHERE D.DEPTNO *= E.DEPTNO
```

RIGHT OUTER JOIN: At this joining concept, Outer Join operator is placed on RIGHT side of joining condition, it will extract matched and unmatched rows from the table which is placed on RIGHT side of Joining condition and substitutes NULL Values for the columns of a table which is on LEFT side of the Joining Condition.

Syntax:

```
SELECT * FROM TABLE1 , TABLE2
 WHERE TABLE1.COLUMN1 =* TABLE2.COLUMN2

SELECT EMPNO, ENAME, JOB, SAL, D.DNAME
  FROM EMP E , DEP D WHERE E.DEPTNO =* D.DEPTNO

SELECT EMPNO, ENAME, JOB, SAL, D.DNAME
  FROM EMP E , DEP D WHERE D.DEPTNO =* E.DEPTNO
```

ANSI JOINS: This are implemented by using key words used at FROM Clause of SELECT STATEMENT.

```
SELECT TABLE1.COLUMN1,TABLE2.COLUMN2
  FROM TABLE1 [ALIAS] < JOIN_TYPE > TABLE2 [ALIAS]
    ON <JOIN_TYPE> -- JOINING CONDITION
TABLE3[ALIAS] ON ---- JOINING CONDITION
```

INNER JOIN OR JOIN :

EQUI JOIN

```
SELECT E.*,D.* FROM EMP E INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
```

NON EQUI JOIN :

```
SELECT E.*, S.* FROM EMP E INNER JOIN SALGRADE S
ON SAL BETWEEN LOSAL AND HISAL
```

EQUI & NON EQUI JOIN:

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

SELF JOIN:

```
SELECT M.EMPNO,M.ENAME,M.MGR,E.ENAME FROM EMP E INNER JOIN EMP M
    ON E.EMPNO = M.MGR
```

CROSS JOIN:

```
SELECT EMP.*, DEPT.* FROM EMP CROSS JOIN DEPT

SELECT EMP.*, DEPT.*, SALGRADE.*
  FROM EMP CROSS JOIN DEPT CROSS JOIN SALGRADE
```

OUTER JOINS

LEFT OUTER JOIN:

```
SELECT E.*, D.* FROM EMP E LEFT OUTER JOIN DEPT D
    ON E.DEPTNO = D.DEPTNO
```

RIGHT OUTER JOIN:

```
SELECT E.* , D.* FROM EMP E RIGHT OUTER JOIN DEPT D
    ON E.DEPTNO = D.DEPTNO
```

FULL OUTER JOIN: It is a combination of LEFT and RIGHT Outer joins.

NESTED QUERIES / SUB QUERIES

- A Query which is written with the other Query refers to NESTED Query.
- A Query which allows to write other query is called Outer Query or Main Query.
- A Query which is written in main query is called INNER or SUB-QUERY

NESTED Query always gets executed in the following ways

- First Executes INNER QUERY
- Then Executes OUTER QUERY

****SQL Server supports to write 32 Sub Queries where as in ORACLE it supporsts up to 255 Sub Queries.**

CLASSIFICATION OF SUB QUERIES

Based on Usage

- Normal Sub Query
- Coorelated Sub Query

Based on Execution

- Single Column – Single Row
- Single Column – Multiple Rows
- Multiple Columns – Single Rows (SQL Server Doesnot supports)
- Multiple Columns – Multiple Rows (SQL Server Doesnot supports)

A SUB-QUERY can be wriiten

- At WHERE Clause
- At HAVING Clause
- At FROM Clause ----- called as INLINE VIEW
- As an Expression ---- called as SCALAR SUB QUERY
- Under DML Queries

SUB-QUERY AT WHERE CLAUSE

- Select as Sub Query can be written at Where Clause of Outer Query.
- Output of Sub Query is never displayed
- Output of Sub Query will be given to OUTER Query for Comparision, Based on It OUTER Query displaces the Output

```
SELECT * FROM TABLE -- MAIN / OUTER QUERY WHERE EXPRESSION OPERATOR (SELECT
FROM TABLE2 WHERE CONDITION)--INNER/SUBQUERY
```

Q) Display EMPNO, ENAME, JOB, SAL, DEPTNO of that Employee who is being paid by Max Salary.

```
SELECT EMPNO, ENAME, JOB, SAL, DEPTNO
  FROM EMP WHERE SAL = ( SELECT MAX(SAL) FROM EMP)
```

****Single row and Single Column.**

Q) Display EMPNO, ENAME, JOB, SAL of those Employees whose salary is more than Average Salary of all Employees.

```
SELECT EMPNO, ENAME, JOB, SAL
  FROM EMP WHERE SAL > ( SELECT AVG (SAL) FROM EMP)
```

Q) Display Ename, Job of those Employees who are working on a same job as of TURNER.

```
SELECT ENAME, JOB
  FROM EMP WHERE JOB = ( SELECT JOB FROM EMP WHERE ENAME = 'TURNER')
```

Q) Display ENAME,SAL of those Employees whose salary is more than average salary of Deptno 20.

```
SELECT ENAME , SAL
  FROM EMP WHERE SAL > ( SELECT AVG(SAL) FROM EMP WHERE DEPTNO = 20)
```

Q) Display ENAME,JOB,SAL of those Employees who are working on a same job as of “Allen” and Salary is more than Salary of “Miller”.

```
SELECT ENAME, JOB, SAL
  FROM EMP WHERE JOB = ( SELECT JOB FROM WHERE ENAME = 'ALLEN')
   AND SAL > ( SELECT SAL FROM EMP WHERE ENAME = 'MILLER')
```

Q) Display Second Max Salary.

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

SELECT ENAME, EMPNO, JOB, SAL , DEPTNO FROM EMP
 WHERE SAL = (SELECT MAX(SAL) FROM EMP
 WHERE SAL < (SELECT MAX(SAL) FROM EMP))
```

Q) Write a Query to display First 3 Max Salaries of Employees.

```
SELECT EMPNO, ENAME, SAL FROM EMP WHERE SAL >= (SELECT MAX(SAL) FROM EMP)
 WHERE SAL < (SELECT MAX(SAL) FROM EMP
 WHERE SAL < (SELECT MAX(SAL) FROM EMP)))
```

Q) Display EMPNO, ENAME, JOB, DEPTNO of those Employees who are working at ACCOUNTING, SALES Departments.

```
SELECT EMPNO, ENAME, JOB, DEPTNO FROM EMP WHERE DEPT IN ( SELECT DEPTNO
FROM DEPT WHERE DNAME IN ('ACCOUNTING', 'SALES'))
```

****Single Column and Multiple rows**

Q) Display all those Employees who are managers.

```
SELECT EMPNO,ENAME,JOB,SAL,DEPTNO
  FROM EMP WHERE EMPNO IN (SELECT DISTINCT MGR FROM EMP
 WHERE MGR IS NULL)
```

Q)Dislay ENAME, JOB, SAL, DEPTNO of those Employees whose salary is more than Max Sal of Sales Department.

```
SELECT EMPNO,ENAME,JOB,SAL
  FROM EMP WHERE SAL > ( SELECT MAX(SAL) FROM EMP
 WHERE DEPTNO = (SELECT DEPTNO FROM DEPT
 WHERE DNAME = 'SALES'))
```

Q) Display Empno,Ename,Job,Sal,Deptno of those Employees who is being paid by Minimum Salary in Deptno 10.

```
SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP
 WHERE SAL = (SELECT MIN(SAL) FROM EMP WHERE DEPTNO =10) AND DEPTNO =10
```

SUB-QUERY AT HAVING CLAUSE

- Select as Sub Query can be written at Having Clause of Outer Query.
- Output Writtened by Sub Query can be used for Comparing the Data at Outer Query, based on it Outer Query displays the Output.

Q) Display DEPTNO & Average Salary of those departments whose average salary is more than the average salary of all Employees.

```
SELECT DEPTNO, AVG(SAL) FROM EMP
HAVING AVG(SAL) > (SELECT AVG(SAL) FROM EMP GROUP BY DEPTNO)
```

SUB QUERY AT FROM CLAUSE { INLINE VIEW }

- Select as Sub Query can be written as FROM Clause of OUTER Query
- Output returns the sub query will be provided as the input for OUTER Query.
- It is must to have an Alias name for the output generated by a sub query
- Column Alias in Sub Query becomes columns for outer query where they can be used at different clause of outer query.

Syntax:

```
SELECT ........ FROM ( SELECT ....... FROM TABLE) ALIAS_NAME

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

```
SELECT EMPNO, ENAME,PAY,DEPTNO FROM ( SELECT EMPNO,ENAME,DEPTNO,SAL PAY
  FROM EMP) WHRE PAY > 2500

SELECT EMPNO,ENAME,SAL,DRNK FROM ( SELECT EMPNO,ENAME,SAL, DENSE_RANK()
  OVER (ORDER BY SAL DESC) DRNK FROM EMP) E WHERE DRNK > 3
```

Q) Write a Query to display First N Maximum Salary earning Employee Details.

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

Q) Write a Query to display Nth Maximum Salary Earning Employees.

```
SELECT EMPNO,ENAME,SAL,DRNK FROM ( SELECT EMPNO,ENAME,SAL, DENSE_RANK()
  OVER (ORDER BY SAL DESC) DRNK FROM EMP) E WHERE DRNK = 3
```

Q) Write a query to display first 5 records of EMP Table.

```
SELECT * FROM ( SELECT EMPNO,ENAME,SAL, ROW_NUMBER()
  OVER (ORDER BY EMPNO) RN FROM EMP) E WHERE RN <= 5
```

Q) Write a query to display all those Employees whose record position at odd Number

```
SELECT * FROM ( SELECT EMPNO,ENAME,SAL, ROW_NUMBER()
  OVER (ORDER BY EMPNO) RN FROM EMP) E WHERE RN %2 =1
```

Q) Write a query to display all thoose Employees whose record position at even numbers

```
SELECT * FROM ( SELECT EMPNO,ENAME,SAL, ROW_NUMBER()
  OVER (ORDER BY EMPNO) RN FROM EMP) E WHERE RN %2 = 0
```

Q) Write a query to display every Forth Record of EMP Table.

```
SELECT * FROM ( SELECT EMPNO,ENAME,SAL, ROW_NUMBER()
  OVER (ORDER BY EMPNO) RN FROM EMP) E WHERE RN %4 =0
```

SUB-QUERY AS AN EXPRESSION {SCALAR SUB QUERY}

Select as Sub Query can be written as an expression in outer query, where it returns the output of only one column, Hence it is named as SCALAR SUB QUERY.

Syntax:

```
SELECT EXPRESSION1, EXPRESSION2.....,(SELECT EXPRESSION .....) [ALIAS
NAME]..... FROM ......
```

Q) Display Ename, Job, Salary, Dept Name of all the Employees.

```
SELECT ENAME,JOB,SAL (SELECT DNAME FROM DEPT
 WHERE DEPTNO = EMP.DEPTNO) DN FROM EMP
```

SOME/ANY & ALL OPERATORS

SOME/ANY:

- This Operator will allow a Sub Query to written multiple rows even though in OUTER Query Condition is made by using relational operators.
- It works like “OR” LOGICAL OPERATOR.
- It Can be used with all RELATIONAL OPERATORS [> ANY , < ANY, = ANY, !=ANY , >= ANY, <= ANY]

```
SELECT * FROM EMP WHERE SAL < ANY (SELECT DISTINCT SAL FROM EMP
 WHERE DEPTNO = 20)
```

ALL :

- This Operator also allows a sub query to written muliple rows even though in Outer Query condition is made using relational operators.
- It works like “AND” LOGICAL OPERATOR.
- It can also be used with ALL Relational Operators.

```
SELECT * FROM EMP WHERE SAL < ALL (SELECT DISTINCT SAL FROM EMP
 WHERE DEPTNO =20)
```

CORRELATED SUBQUERIES :

- This Queries provides different execution to Nested Queries i.e First it executes OUTER Query then executes INNER Query.
- Nested Query is called as UNI-DIRECTIONAL Query
- Correlated Query is called as BI-DIRECTIONAL Query
- Correlated queries gets executed in the following way First executes OUTER query retriveing only one ROW(called as CANDIDATE ROW) at a time and that ROW is given to INNER Query for processing after procesing INNER Query returns Output and will be given to OUTER Query for comparision, based on it OUTER Query displays the Output.
- This Queries are identified when outer queris table name is provided with the Alias name and that alias Name is been used at Sub-Query.

Q) Display EMPNO,ENAME,JOB,SAL,DEPTNO of those Employees whose salary is more than AVERAGE SALARY of respective Departments.

```
SELECT EMPNO,ENAME,JOB,SAL,DEPTNO FROM EMP E
 WHERE SAL > ( SELECT AVG(SAL) FROM EMP WHERE DEPTNO = E.DEPTNO )
```

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7839	KING	PRESIDENT	5000	10
2	7902	FORD	ANALYST	3000	20
3	7788	SCOTT	ANALYST	3000	20
4	7566	JONES	MANAGER	2975	20
5	7698	BLAKE	MANAGER	2850	30
6	7499	ALLEN	SALESMAN	1600	30

Q) Dispay the Employee Details of the First N Maximum Salaries

```
SELECT EMPNO, ENAME, JOB, SAL , DEPTNO FROM EMP E
 WHERE 3 > ( SELECT COUNT(DISTINCT SAL) FROM EMP WHERE SAL > E.SAL)
```

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7566	JONES	MANAGER	2975	20
2	7788	SCOTT	ANALYST	3000	20
3	7839	KING	PRESIDENT	5000	10
4	7902	FORD	ANALYST	3000	20

Q) Display the Employee Details of Nth Maximum Salary.

```
SELECT EMPNO, ENAME, JOB, SAL , DEPTNO FROM EMP E
 WHERE 3 = ( SELECT COUNT(DISTINCT SAL) FROM EMP WHERE SAL > E.SAL)
```

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7698	BLAKE	MANAGER	2850	30

****To find Nth Maximum Salary condition should be (N-1)**

Q) Display the Employee Details of Forth Maximum Salary.

```
SELECT EMPNO, ENAME, JOB, SAL , DEPTNO FROM EMP E
 WHERE 4 = ( SELECT COUNT(DISTINCT SAL) FROM EMP WHERE SAL > E.SAL)
```

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7782	CLARK	MANAGER	2450	10

Q) Display those Employee Details who are being paid by minimumm salaries in their respective Departments.

```
SELECT EMPNO, ENAME, JOB, SAL , DEPTNO FROM EMP E
 WHERE SAL = ( SELECT MIN(SAL) FROM EMP WHERE DEPTNO = E.DEPTNO)
```

	EMPNO	ENAME	JOB	SAL	DEPTNO
1	7934	MILLER	CLERK	1300	10
2	7369	SMITH	CLERK	800	20
3	7900	JAMES	CLERK	950	30

EXISTS / NOT EXISTS OPERATOR

EXISTS:

- It Returns Boolean value ie.. True or False
- If Condition at inner query is satisfied than it will written True else written with False

Q) Display DEPTNO, DNAME of those Department's where atleast one Employe is working.

```
SELECT DEPTNO, DNAME FROM DEPT D
 WHERE EXISTS ( SELECT 1 FROM EMP WHERE DEPTNO = D.DEPTNO )
```

	DEPTNO	DNAME
1	10	ACCOUNTING
2	20	RESEARCH
3	30	SALES

NOT EXISTS:

- This operator also writtens Boolean Value i.e. TRUE or FALSE
- If Condition at INNER Query is FALSE then it returns TRUE

Q) Display DEPTNO, DNAME of those Department's where no Employee is working.

```
SELECT DEPTNO, DNAME FROM DEPT D
 WHERE NOT EXISTS (SELECT 1 FROM EMP WHERE DEPTNO = D.DEPTNO)

OR

SELECT DEPTNO,DNAME FROM DEPT
 WHERE DEPTNO = (( SELECT DEPTNO FROM DEPT )
                   EXCEPT (SELECT DISTINCT DEPTNO FROM EMP))
```

	DEPTNO	DNAME
1	40	OPERATIONS

CREATING A NEW TABLE FROM EXISTING TABLE:

- When a new table is created from existing table it will support to copy METADATA + ACTUAL DATA
- It also supports to copy only Metadaa when condition is false
- Column Aliases specified for the Columns of existing table becomes Columns in new tables
- Constraints of an existing table columns are never copied to new table

Syntax :

```
      SELECT * INTO <NEW_TABLE_NAME> FROM < EXISTING_TABLE >

SELECT * INTO EMP1 FROM EMP
SELECT * INTO EMP2 FROM EMP WHERE 1 = 2

SELECT EMPNO ECODE, ENAME EN, SAL PAY, SAL*12 ASAL, DEPTNO
  INTO EMP3 FROM EMP
```

	ECODE	EN	PAY	ASAL	DEPTNO
1	7369	SMITH	800	9600	20
2	7499	ALLEN	1600	19200	30
3	7521	WARD	1250	15000	30
4	7566	JONES	2975	35700	20
5	7654	MARTIN	1250	15000	30

DATA MANIPULATION LANGUAGE:

- Insertion of New Rows, Modification to existing Rows, Removing Unwanted Rows collectively known as DATA MANIPULATION LANGUAGE.
- It Includes the Following Commands INSERT, UPDATE, DELETE, MERGE

INSERT:

- This DML Command is used to Insert a row into an existing table , it provides its working in 3 ways.
- Inserting ONE Row at a time in ONE Table.
- Inserting MULTIPLE Rows at a time in ONE Table.
- Inserting MULTIPLE Rows at a time in MULTIPLE Tables.

INSERT ..... ONE ROW ..... ONE TABLE

- When One Row is inserted at a time in One Table it supports to store data of all Columns or Partial Columns
- When Datais inserted for required Columns, The Other Columns will be stored by NULL Values

Syntax:

```
INSERT [ INTO ] <TABLE_NAME>[ COLUMN_ALIAS ]
      VALUES ( LIST_OF_VALUES )
```

```
INSERT DEPT VALUES (50,'HR','HYD')
INSERT DEPT VALUES (60,'ADMIN')
INSERT DEPT VALUES (70,'ECE',NULL)

INSERT DEPT (DEPTNO,DNAME)
            VALUES (80,'FINANCE'),
            VALUES (90,'ARTS'),
            VALUES (95,'TEST')
```

****Using Values in More than once in INSERT Command is valid in 2008 Version of SQL Server.**

INSERT ..... MULTIPLE ROWS ..... IN ONE TABLE

Select as Sub-Query can be written under Insert Query, Output returned by Sub Query can be used by INSERT Query to INSERT the Rows into an Existing Table.

Syntax:

```
INSERT [ INTO ] <TABLE_NAME> [(COLUMNS_LIST)] < SELECT QUERY >
```

```
INSERT EMP2 SELECT * FROM EMP

INSERT EMP2 ( EMPNO, ENAME, JOB, SAL) SELECT (EMPNO, ENAME, JOB, SAL)
  FROM EMP

INSERT EMP2 ( EMPNO, ENAME, JOB, SAL) SELECT (EMPNO, ENAME, JOB, SAL)
  FROM EMP WHERE DEPTNO =10
```

UPDATE :

This DML Command is used to modify existing Data with User Required new Data.

Syntax:

```
UPDATE < TABLE_NAME >
   SET COLUMN1 = VALUE/EXPRESSION, COLUMN2 = VALUE/EXPRESSION
 WHERE = CONDITIONS
```

Q) Update the Salary of SMITH from 800 to 1800.

```
UPDATE EMP SET SAL = 1800 WHERE ENAME = 'SMITH'
```

Q) Update the JOB, SAL of MARTIN from SALESMAN to CLERK and 1250 to 2000.

```
UPDATE EMP SET JOB = 'CLERK', SAL = 2000 WHERE ENAME = 'MARTIN'
```

Q) Update The Salary of Department 30 with an increament of 12%.

```
UPDATE EMP SET SAL = SAL*12 WHERE DEPTNO = 30
```

Q) Update the Salary of James with an increament of 1000 and provided the Comm as 10% of Salary.

```
UPDATE EMP SET SAL = SAL+1000, COMM = (SAL+1000)*10/100.0
 WHERE ENAME = 'JAMES'
```

Q) Update the Salary of ALLEN with an increament of 1000, Sal of JAMES with 500 and other Employees sal should remain unchanged.

```
UPDATE EMP
   SET SAL = CASE ENAME
                WHEN 'ALLEN' THEN SAL + 1000
                WHEN 'JAMES' THEN SAL + 500
             ELSE SAL
             END
 WHERE ENAME IN( 'ALLEN','JAMES')
```

- When Multiple Colums are used with Update Command all the Columns get updated at a time.
- Single column cannot be used for more than once
- When Single Column has modification with different Values we need to use CASE Expression

SUB-QUERY UNDER UPDATE QUERY

SELECT as Sub-Query can be written under update Query OUTPUT returned by Sub-Query can be used for Updating the Data in Outer Query.

Q) Update the Salary of the SMITH with the salary of MILLER.

```
UPDATE EMP
   SET SAL = ( SELECT SAL FROM EMP WHERE ENAME = 'MILLER' )
 WHERE ENAME = 'SMITH'
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	1300	NULL	20
2	7934	MILLER	CLERK	7782	1982-01-23 00:00:00.000	1300	NULL	10

Q) Update the Salary of 30 Department with the Average Salary of 20 Department

```
UPDATE EMP
   SET SAL = ( SELECT AVG( SAL ) FROM EMP WHERE DEPTNO =20 )
 WHERE DEPTNO = 30
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17 00:00:00.000	1300	NULL	20
2	7566	JONES	MANAGER	7839	1981-04-02 00:00:00.000	2975	NULL	20
3	7788	SCOTT	ANALYST	7566	1982-12-09 00:00:00.000	3000	NULL	20
4	7876	ADAMS	CLERK	7788	1983-01-12 00:00:00.000	1100	NULL	20
5	7902	FORD	ANALYST	7566	1981-12-03 00:00:00.000	3000	NULL	20
6	7900	JAMES	CLERK	7698	1981-12-03 00:00:00.000	2275	NULL	30
7	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.000	2275	0	30
8	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.000	2275	1400	30
9	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.000	2275	NULL	30
10	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.000	2275	300	30
11	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.000	2275	500	30

DELETE :

It is used to Delete One or More rows at a time from an existing table.

Syntax :

```
DELETE FROM < TABLE_NAME > [ WHERE CONDITIONS ]
```

Q) Delete all the records of Dept Table.

```
DELETE FROM DEPT
```

Q) Delete all the Employees who are working as CLERK.

```
DELETE FROM EMP WHERE JOB = 'CLERK'
```

Q) Delete all those Employees who are earning Commission.

```
DELETE FROM EMP WHERE COMM IS NOT NULL
```

SUB-QUERY UNDER DELETE QUERY

SELECT as SUB-QUERY can be written under DELETE Query Output returned by SUB-QUERY can be used for deleting the records through Outer Query.

Syntax:

```
DELETE FROM < TABLE_NAME > WHERE EXPRESSION OPERATOR ( SELECT QUERY )
```

Q) Delete all those Employees whose Salary is more than Average Salary of all Employees.

```
DELETE FROM EMP WHERE SAL > ( SELECT  AVG(SAL) FROM EMP )
```

Q) Delete all those Employees who are working on a same Job as of SMITH.

```
DELETE FROM EMP WHERE JOB = (SELECT JOB FROM EMP WHERE ENAME = 'SMITH')
```

MERGE :

- It is introduced in SQL SERVER 2008
- It is called as an ETL Command (Extracting, Transformation, Loading)
- It is used to Join the data from 2 Tables into a Single Table
- It is used to Execute INSERT, UPDATE, DELETE commands simmaltenously
- It works based on Four Clauses USING, ON , WHEN MATCHED, WHEN NOT MATCHED

- WHEN MATCHED, WHEN NOT MATCHED Clauses are Flexible in usage any number of times can be used.
- Usage of Semi-Colon at the end of the Query is compulsory.

Syntax :

```
MERGE INTO < TABLE_1_NAME > [TABLE_1_ALIAS]
 USING < TABLE_2_NAME > / < SELECT QUERY > [ TABLE_2_ALIAS ]
    ON  JOINING CONDITION
  WHEN MATCHED [ FILTERING CONDITIONS ] THEN
UPDATE / DELETE COMMANDS
  WHEN NOT MATCHED [ FILTERING CONDITIONS ] THEN
INSERT [ COLUMNS_LIST ] VALUES [ LIST_OF_VALUES ] ;
```

Example:

```
MERGE INTO EMP E1
 USING (SELECT EMPNO, ENAME, JOB, SAL, SAL*12 ASAL FROM EMP) E2
    ON (E1.EMPNO = E2.EMPNO)
  WHEN MATCHED THEN
UPDATE SET E1.SAL = E2.SAL
  WHEN NOT MATCHED THEN
INSERT VALUES (E2.EMPNO,E2.ENAME,E2.JOB,E2.ASAL);
 MERGE INTO EMPB E1
 USING (SELECT EMPNO, ENAME, JOB, SAL, SAL*12 ASAL FROM EMP)E2
    ON (E1.EMPNO = E2.EMPNO)
  WHEN MATCHED AND E1.JOB = 'CLERK' THEN
UPDATE SET E1.SAL = E2.SAL
  WHEN MATCHED AND E1.JOB = 'SALESMAN' THEN
DELETE
  WHEN NOT MATCHED AND E2.JOB = 'ANALYST' THEN
INSERT VALUES( E2.EMPNO,E2.ENAME,E2.JOB,E2.ASAL);
```

TRANSACTION CONTROL LANGUAGE

- This Language is used to make a transaction either Permanent of Supports to cancel the Transaction.
- It Includes COMMIT, ROLLBACK, SAVE Commands

TRANSACTION:

- It is defined as small logical unit which contains set of operations.
- Transaction will works on ACID Properties
  - A – Atomicity
  - C – Consistency
  - I – Isolated
  - D – Durability
- In SQL Server 2005 Transactio will start with a Key Word " BEGIN TRAN / TRANSACTION " [NAME]
- By default set of operations associated to a transaction or temporary.

Syntax :

```
BEGIN TRANSACTION
      < STATEMENTS >
```

COMMIT :

- This TCL Command is used to Make a transaction (SET OF OPERATIONS) Permanent in a DATABASE.

- When multiple isolate transactions are created requires equal number of COMMITS to make them permanent in a DATABASE.

ROLLBACK :

- This TCL Command is used to Cancel the Complete Transaction or Partial Transaction
- When isolated Multiple transactions are created in a Singel Session.
- A Single ROLLBACK command would cancel all the transactions.

```
ROLLBACK [ TRAN / TRANSACTION ] NAME
```

SAVE :

- This TCL Command is used to create a Sub-Transactions under a Main Transcation. The Created Sub Transaction can be cancelled partialy.
- Random Selection from Cancellaiton of Sub Transaction is not allowed

```
SAVE [ TRAN / TRANSACTION ] NAME
```

Example:

```
BEGIN TRAN
  DELETE FROM EMP WHERE DEPTNO = 30
  SAVE TRAN F
    INSERT DEPT VALUES (.....)
    UPDATE DEPT SET COL = VAL WHERE DEPTNO = 10
  SAVE TRAN S
    DELETE FROM SALGRADE
    DELETE FROM EMP
```

Three Scenarios for above example.

First	Second	Third
ROLLBACK	ROLLBACK TRAN S	ROLLBANK TRAN F
	ROLLBACK TRAN F	ROLLBACK
	ROLLBACK	

When a session contains a transaction it cannot be closed directly , if it is trying closed it confirms for COMMIT and ROLLBACK of Transactions.

DATA DEFINATION LANGUAGE :

- This language is used to Create, Modify and Drop DATABASE Objects ie. Tables, Views, Indexes, Synonms, Stored Procedures , Stored Functions, Stored Triggers.
- This language includes the following Commands CREATE, ALTER, TRUNCATE, DROP
- In SQL Server 2005 if DDL Commands are associated to Transaction then the Operation is Temporary.

REPRESENTATION OF DATA

Data in SQL Server gets represented based on three Key Points DATANAME, DATATYPE, DATASIZE.

DATA NAME : Often Called Column Name , Identifier or Field or Attribute. It is the one which gives Identification to Data.

Rules to be followed before using Column Names :

- Column Name can be Minimum of 1 Character and Maximum of 127 Characters.

- Table Should contain always Unique Column Names.
- Column Names used in 1 Table the same can be used in other tables.
- Column Names cant be enclosed with the Blank Spaces and Special Characters.
- In order to have Blank Spaces and Special Characters then it should be enclosed in Square Brackets.

DATA TYPE & DATA SIZE

- Data Type explains regarding type of data to be stored in a column.
- SQL Server 2005 provides the following Data Types for the storage of data with respect to structured and unstructured data.
- Structured Data refers to the storage of DATA directly.
- Unstructured Data refers to the storage of Data which is external Data.

INTEGER DATA [EXACT NUMBERS]

```
INT         4 BYTES
SMALLINT    2 BYTES [ -32768 to 32767 ]
BIGINT      8 BYTES
TINYINT     1 BYTES [ 0 to 255 ]
DECIMAL     (PREC [SCALE])
NUMERIC     (PREC [SCALE])
MONEY       8 BYTES
SMALLMONEY  4 BYTES
```

REAL NUMBERS: (APPROXIMATES, NUMBERS)

```
FLOAT       VARIES
REAL        4 BYTES
```

DATE AND TIME :

```
DATETIME          8 BYTES { 1ST JAN 1753 to 31ST DEC 9999 }
SMALLDATETIME     4 BYTES { 1ST JAN 1900 to 06TH JUN 2079 }
```

STRING DATATYPE:

`CHAR(SIZE)`

- Supports to store any type data.
- Default size it will take as ONE.
- Provides Fixed Memory / Static Memory which leads to wastage of Memory.
- It Stores data using “ASCII” Code Conept.
- Minimum can be 1 BYTE Maximum of 8000 BYTES.

`NCHAR( SIZE )`

- Supports to store any type data
- Default size it will take as ONE
- It Stores data using “UNICODE” Concept
- Size can be specified Minimum 1 BYTE Maximum 4000 BYTES

`VARCHAR( SIZE )`

- Supports to store any type data
- Provides variable Length Memory, Dynamic Memory which results to saving of Memory.
- It Stores data using “ASCII” Code Conept

- Size can be specified Minimum 1 BYTE Maximum 8000 BYTES

```
NVARCHAR ( SIZE )
```

- Supports to store any type data
- Provides variable Length Memory, Dynamic Memory which results to saving of Memory.
- It Stores data using "UNICODE" Code Conept
- Size can be specified Minimum 1 BYTE Maximum 8000 BYTES

UNSTURCTURED DATA

```
BINARY
VARBINARY
IMAGE
```

XML – Introduced from 2005 to store data in XML Format,
Memory is provided upto 2GB.

CREATING TABLE AS DATABASE OBJECT

Table is a logical Sturcture which provides data in the form of ROWS & COLUMNS.

Syntax:

```
CREATE TABLE < TABLE_NAME > (
      COLUMN1 COLUMN_DEFINATION,
      COLUMN2 COLUMN_DEFINATION,
      COLUMN3 COLUMN_DEFINATION.....)
```

Note:

- A DATABASE can contain TWO BLLION Tables.
- A Table can have Minimum of One Column and Maximum of 1024 Columns.
- No Limit for Rows.
- All the DATABASE Objects by default gets associated to predefined Schema i.e "dbo" [DATABASE OWNER]
- DATABASE Objects should be unique in Schema.
- Table Name can be of Minimum One Character and Maximum of 127 Characters.

Tables are classified in 3 Types

System Tables -------- Master DB's ---------- 41 Tables
User Defined Tables ------- User Defined Database ----------2Billion Tables
Temporary Tables ------- TEMPDB ----------Unlimited

TYPES OF COLUMNS

Column in SQL Server 2005 are classified in following types.

Normal Column: Column Associated by DATA TYPE & DATA SIZE.

```
RNO INT, SNAME VARCHAR (10)
```

Computed Column: Column Assocaited by Mathematical Expression.

```
AMT AS ( PRICE * QTY ), ASAL AS ( SAL * 12)
```

Points to be remember:

- This column is introduced from 2005.
- DATA TYPE of computed column is based on Datatype of Those Columns which are used in Mathematical Expression.
- One Computed column cannot be under other Computed Column.
- Explicit data is not allowed for computed columns.
- A Table can have more than ONE Computed Columns.

IDENTITY COLUMN

- A table can have only ONE IDENTITY Column.
- IDENTITY Column will generate sequence of numbers automatically by default starts from One and increaments by One.
- By Default IDENTITY Column will not accpet Explicit Values.
- IDENTITY Column can be specified by User Defined Start (SEED) and Increament Value.

Syntax:

```
IDENTITY ( SEED, INCREAMENT )
SNO INT IDENTITY
```

****By default it starts from 1 and it will increament by 1**

```
SNO INT IDENTITY (101,4)
```

****It starts from 101 and it will increament by 4**

- In order to store explicit values for IDENTITY COLUMN Following SET Command should be used.

```
SET IDENTITY_INSERT < TABLE_NAME > ON / OFF [ DEFAULT IS OFF ]
```

- When Explicit Values are provided for identity colum it is must to mention that column name in the Column List.
- IDENTITY COLUMN will allows duplicates.
- When Explicit Values are stored in IDENTITY COLUMN, later when implicit values are stored it would be generate the sequence of numbers based on the biggest value of that column.
- IDENTITY COLUMN can be associated with the following data types i.e INT,BIGINT,SMALLINT,TINYINT,DECIMAL OR NUMERIC with SCALE

@@IDENTITY - It returns the Last Value stored into IDENTITY COLUMN (Related to Last Table)
It works at session level.

SCOPE_IDENTITY - It returns the Last Value stored into IDENTITY COLUMN.

```
SELECT SCOPE_IDENTITY()
```

IDENT_CURRENT - This function returns the last IDENTITY COLUMN value which is stores in a specific table.

```
SELECT IDENT_CURRENT( < TABLE_NAME > )
```

SYS.IDENTITY_COLUMNS - This System Table also supports to get the last value of IDENTITY COLUMN of any table.

```
SELECT LAST_VALUE FROM SYS.IDENTITY_COLUMNS
 WHERE [OBJECT_ID] = OBJECT_ID( < TABLE_NAME > )
```

IDENT_SEED (< TABLE_NAME >) - It returns the start value of the IDENTITY COLUMN created at table.

```
SELECT IDENT_SEED ('PRODUCT')
```

IDENT_INCR (< TABLE_NAME >) - It returns the increamented value of the IDENTITY COLUMN created at table.

```
SELECT IDENT_INCR ('PRODUCT')
```

To restart the sequence of Numbers for IDENTITY COLUMN an option called CHECKIDENT of DBCC (DATABASE CONSISTENCY CHECKER) Command.

```
DBCC CHECKIDENT ( <TABLE_NAME>, RESEED , VALUE )
```

Example:

```
CREATE TABLE PRODUCT (  SNO INT IDENTITY, PCODE INT,  PNAME VARCHAR(10),
QTY INT, PRICE FLOAT,   AMT AS (QTY * PRICE) )

INSERT INTO PRODUCT VALUES (201,'X',10,15)
INSERT INTO PRODUCT VALUES (202,'Y',10,15)
INSERT INTO PRODUCT VALUES (203,'Z',10,15)
INSERT INTO PRODUCT VALUES (204,'A',10,15)
INSERT INTO PRODUCT VALUES (204,'B',10,15)

SET IDENTITY_INSERT PRODUCT ON

INSERT INTO PRODUCT (SNO,PCODE,PNAME,QTY,PRICE)
VALUES (6,205,'C',10,15)

SELECT @@IDENTITY

SELECT SCOPE_IDENTITY()

SELECT IDENT_CURRENT('PRODUCT')

SELECT LAST_VALUE FROM SYS.IDENTITY_COLUMNS
 WHERE [OBJECT_ID] = OBJECT_ID('PRODUCT')

DBCC CHECKIDENT('PRODUCT',RESEED,3)
```

Q) Create a Table Called STUDENT which contains ROLLNO, NAME, MPC MARKS.
Calculate for total and average marks.

Using Computed Column

```
CREATE TABLE STUDENT (  ROLLNO INT , STNAME VARCHAR(10),
      MATHS INT,  PHYSICS INT, CHEMISTRY INT,
      TOTAL AS MATHS*PHYSICS*CHEMISTRY,
      AVGM AS ((MATHS*PHYSICS*CHEMISTRY)/3.0))
```

Using UPDATE Method

```
CREATE TABLE STUDENT (  ROLLNO INT , STNAME VARCHAR(10),
      MATHS INT,  PHYSICS INT, CHEMISTRY INT,
      TOTAL INT, AVGM FLOAT)
```

```
UPDATE STUDENT
   SET TOTAL = MATHS*PHYSICS*CHEMISTRY,
       AVGM =((MATHS*PHYSICS*CHEMISTRY)/3.0))
```

Q) Create a table EMP which contains ECODE, ENAME & GENDER, UPDATE Gender by changing the data to MALE to FEMALE and FEMALE to MALE.

```
UPDATE EMP
   SET GENDER = CASE GENDER
       WHEN 'MALE' THEN 'FEMALE'
       WHEN 'FEMAE' THEN 'MALE'
   END
```

MODIFYING EXISTING TABLE

- Existing Table can be modified by using ALTER TABLE Command
- It supports to add new columns, modify data types and data size of existing columns and supports to drop the columns
- ALTER TABLE Command provides three options ADD, ALTER COLUMN, DROP.
- To add the columns – Normal Columns, Computed Columns, Identity Columns can be added into an existing table.

Syntax:

```
      ALTER TABLE < TABLE_NAME >
        ADD COLUMN1 COLUMN1_DEFINATION, COLUMN2 COLUMN2_DEFINATION .......

ALTER TABLE EMP ADD SNO INT IDENTITY, EXPR INT, ASAL AS ( SAL *12)
```

Modifying Data Type & Size of existing Table

Data Type and Data Size can be altered in a flexible way if there is no data in a column. It Supports to modify only independent Columns

Syntax:

```
ALTER TABLE < TABLE_NAME > ALTER COLUMN COLUMN_NAME COLUMN_DEFINATION
      ALTER TABLE EMP ALTER COLUMN EMPNO TINYINT
      ALTER TABLE EMP ALTER COLUMN ENAME VARCHAR(20)
```

DROPPING THE COLUMNS:

- One or more columns can be dropped from a table.
- It drops a column including data
- It will drop only independent columns
- It does not allow to drop all the columns of a table (Minimum 1 Column should be there in a table)

Syntax :

```
      ALTER TABLE < TABLE_NAME > DROP COLUMN COLUMN_NAME .....

      ALTER TABLE EMP DROP COLUMN ENAME, SAL, JOB
```

SP_RENAME

- This predefined procedure will support to rename columns and table
- It will suppport to rename only independent columns and tables

Syntax:

```
SP_RENAME ' < TABLENAME.COLUMNNAME > ', ' < NEW_COLUMN_NAME >'

SP_RENAME 'EMP.EMPNO','ECODE'

SP_RENAME ' < OLD_TABLENAME > ', ' < NEW_TABLENAME >'
```

TRUNCATE :

This DDL Command is used to delete the rows from a table

Syntax: `TRUNCATE TABLE < TABLE_NAME >`

DELETE	TRUNCATE
• It is a DML Command	• It is a DDL Command
• Supports to delete the rows on conditional basis	• No Conditional deletion is allowed
• Slow in Execution	• Fast in Execution
• Supports to invoke Triggers	• Does not support
• Identity Column Value will be in continuation even after deleting all rows.	• When a table is truncated Identity Column Values will be resetted to start position(SEED VALUE)

DROP:

This DDL Command is used to drop the DATABASE Objects like tables, views, synonms, indexes, Stored Procedures, Functions, Trigers Etc.

Dropping a Table:

When a table is dropped, objects like constraints, indexes, triggers are automattically dropped.

Syntax: `DROP TABLE < TABLE_NAME >`

WORKING WITH UNSTRUCTURED DATA

Inserting IMAGE

```
CREATE TABLE EIMG (
      ENO INT, ENAME VARCHAR(12) EPIC VARBINARY(MAX) )

INSERT INTO EIMG (ENO,ENAME,EPIC) SELECT 1001,'XYZ', BULK COLUMN
  FROM OPENROWSET (BULK 'C:\TEST.JPG',SINGLE_BLOB) AS EIMG
```

Inserting Word Document

```
CREATE TABLE EDOC (
      ENO INT, ENAME VARCHAR(12) EDOC VARBINARY(MAX) )

INSERT INTO EIMG (ENO,ENAME,EPIC) SELECT 1001,'XYZ', BULK COLUMN
  FROM OPENROWSET (BULK 'C:\TEST.DOC',SINGLE_BLOB) AS DOC
```

WORKING WITH TOP KEYWORD

- This keyword is used to perform operations on TOP rows of a table.
- This keyword in SQL SERVER 2000 was used only with SELECT Command whereas in 2005 it can be used with SELECT,UPDATE,DELETE Commads
- It can retrive the data based on specific number or based on Percentage
- It will restrict the resultset of a Query.

```
SELECT TOP (3) * FROM EMP
SELECT TOP 50 PERCENT * FROM EMP
SET ROWCOUNT 3
```

****To get Top rows with out using TOP Keyword or WHERE Clause, we need to use SET ROWCOUNT Function.**

Q) Display maximum salaries

```
SELECT * FROM EMP WHERE SAL IN (
            SELECT DISTINCT TOP 3 SAL FROM EMP ORDER BY SAL DESC)

UPDATE TOP(3) FROM EMP SET SAL = SAL + 1000

DELETE TOP (3) FROM EMP
```

CREATING TEMPORARY TABLES

- This tables will be provided by memory for temporary at TEMPDB(System DB)
- This tables will work like normal tables.
- This tables will be deleted automatically when the session is closed
- This tables are classifed in 2 Types.
  - Local Temp Table
  - Global Temp Table
- Local – It works at Single Connection and it is identified by " # "
- Global – It works at Multiple Connection and it is identified by " ## "

```
CREATE TABLE #T1 (RNO INT, SNAME VARCHAR(10))

INSERT INTO #T1 VALUES (1,'X')
INSERT INTO #T1 VALUES (2,'Y')
INSERT INTO #T1 VALUES (3,'Z')

SELECT * INTO #D1 FROM DEPT
SELECT * INTO TT1 FROM #T1
SELECT * INTO #T1 FROM #TT

CREATE TABLE ##T1 (RNO INT, SNAME VARCHAR(10))

INSERT INTO ##T1 VALUES (1,'X')
INSERT INTO ##T1 VALUES (2,'Y')
INSERT INTO ##T1 VALUES (3,'Z')
```

****All Temparory Tables are independent.**

DATABASE CONSTRAINTS:

- Constraints refers to rules or conditions

- Data Constraint will make SQL Server to avoid invalid data entry.
- Every Constraint is provided by a unique name, which can be of predefined or user defined.
- To specify User Defined Constraint Name a Keyword called CONSTRAINT is used.
- Advantage of Constraint Name is found when a constraint is disabled/enabled or when the constraint is dropped.
- Constraint is created as DATABASE Object.
- Constraint Names should be always provided with unique names Because it will be checked at Schema Level.

Types of Constriants

- DOMAIN INTEGRITY CONSTRAINTS -- NOTNULL, CHECK, DEFAULT
- ENTITY INTEGRITY CONSTRAINTS -- PRIMARY KEY, UNIQUE
- REFERENTIAL INTEGRITY CONSTRAINTS -- FOREIGN KEY

DOMAIN INTEGRITY CONSTRAINT

These constaints are specified immediately after a Column, It includes NOTNULL, CHECK, DEFAULT.

NOTNULL

- It is not a Constraint.
- It can be provided for any Number of Columns.
- It will not allow to store NULL VALUES, But will allow to store Duplicates

CHECK

- This Constraint is used to specify User Defined Conditions
- Any Number of Columns Can be set with the CHECK Constratint

DEFAULT

- This Constraint is used to specify Default Value for Column.
- When a user does not provide data (INSERT) then it will store Default value.

Example:

```
CREATE TABLE EMP1(
ENO INT NOT NULL, ENAME VARCHAR(10) NOT NULL,
JOB VARCHAR(15) CHECK (JOB IN ('CLERK','MANAGER','OPERATIONS')),
SAL INT CONSTRAINT SAL_CHK CHECK (SAL BETWEEN 15000 AND 20000),
DOJ SMALLDATETIME DEFAULT GETDATE(),
DNO INT CONSTRAINT DF_DNO DEFAULT 20)
```

ENTITY INTEGRITY CONSTRAINT

These Constraints can be specified in two levels.

- Column Level – Immediately after a Column.
- Table / Entity Level – after all the columns
- It Includes PRIMARY KEY, UNIQUE CONSTRAINTS

UNIQUE

- This Constraint will not allow to store duplicate values but can store only ONE NULL Value.
- It Can be used for any number of times

COLUMN LEVEL

```
CREATE TABLE STUDENT
(RNO INT UNIQUE, SN VARCHAR(10) CONSTRAINT SN_UQ UNIQUE)
```

TABLE / ENTITY LEVEL

```
CREATE TABLE STUDENT
(RNO INT , SN VARCHAR(10),UNIQUE(RNO) CONSTRAINT SN_UQ UNIQUE(SN))
```

COMPOSITE UNIQUE KEY

- Multiple Columns set with One UNIQUE Constraint is called Composite Unique Key.
- It is created only at Entity Level or Table Level.
- Composite Unique will allow to store one column with duplicates only when corresponding Column contains Unique Data
- Minimum of two Columns and Maximum of 16 Columns.

Example:

```
CREATE TABLE STUDENT(
RNO INT, SNAME VARCHAR(10), CLASS INT, UNIQUE (RNO,CLASS))
```

PRIMARY KEY: { NOT NULL + UNIQUE }

- Which means it will not allow NULL values and duplicates.
- A table can have only one Primary Key which can be created at Table or Column Level.

COLUMN LEVEL

```
CREATE TABLE STUDENT ( RNO INT PRIMARY KEY, SNAME VARCHAR(10))
```

TABLE LEVEL

```
CREATE TABLE STUDENT ( RNO INT , SNAME VARCHAR(10) PRIMARY KEY (RNO))
```

COMPOSITE PRIMARY KEY

- Multiple Columns set with One Primary key refers to Composite Primary Key.
- It can be created only at table level.
- Behaviour of Composite Primary Key that it will allow One column Store Duplicates only when the corresponding column contains Unique Data.
- Composite Primary Key can be created Minimum of 2 and Maximum of 16 Columns

****Difference in Composite Unique Key and Composite Primary Key – Composite Unique Key will allow NULL Values, Composite Primary Key will not allow NULL Values.**

```
CREATE TABLE STUDENT (
RNO INT, SNAME VARCHAR(10), CLASS INT, PRIMARY KEY (RNO, CLASS))
```

CREATING MULTIPLE CONSTRAINTS ON A SINGLE COLUMN

```
CREATE TABLE EMPLOYEE (
ENO INT PRIMARY KEY,
ENAME VARCHAR(15) NOT NULL UNIQUE,
SAL INT NOT NULL UNIQUE CHECK (SAL BETWEEN 15000 AND 20000))
```

REFEREINTAL INTEGRITY CONSTRAINT

- This Constraint is used to create relationships among tables.

- It supports to create following relationships.
  ONE to ONE and ONE to MANY { DIRECT IMPLEMENTATION }
  MANY to MANY { INDIRECT IMPLEMENTATION }

FOREIGN KEY

- It will allow to store valid values that are present in another Column.
- Foregin Key Column Name and the Column where the data is been checked can be with same name are different names.
- If column names are different it is must that the Data Types and Data Size should be same.
- When Foreign Key Column data gets referred to another column, on that column there should be either a PRIMARY KEY or a UNIQUE Constraint.
- Foreign key is a bidirectional key since
  - At Insertion it works from CHILD to PARENT
  - At Deletion it works from PARENT to CHILD
- Foreign key column and the other column where data is been checked can be in same table or can be in different tables. If they are in same table it is called as SELF REFERENCED TABLE or REFLEXIVE RELATION.
- Foreign key allows NULL Values and Duplicates.
- More than One Foreign Key can be created in a single table.
- Microsoft recommends to create max of 253 Foreign Keys in a single table. (Practically it is possible to create more than 253 Foreign Keys).
- Foreign key in SQL SERVER can be created at column level or table level.(In ORACLE only at Table Level)
- Foreign Key can be extended by the following options
  - ON DELETE CASCADE
  - ON UPDATE CASCADE
  - ON DELETE SET NULL
  - ON UPDATE SET NULL
  - ON UPDATE SET NO ACTION(default)
- When ON DELETE CASCADE is specifed, it will allow to delete the records from parent table directly, the same record existing at child table will be deleted automatically.
- When ON UPDATE CASCADE is specifed it will allow to modify the records from parent table directly, the same record existing at child table will be modified automatically.
- When ON DELETE SET NULL is specified, it will allow to delete the records from parent table directly, the same data existing child table in Foreign key column will be set to NULL.
- When ON UPDATE SET NULL is specified, it will allow to delete the records from parent table directly, the same data existing child table in Foreign key column will be set to NULL .
- Changes made in One table when they are reflected automatically in other table the process is known as data consistency.

Creating One to One Relation Ships

```
CREATE TABLE REGISTER( REGNO INT PRIMARY KEY,
SNAME VARCHAR(10), COURSE VARCHAR(10))

CREATE TABLE RESULT( HTNO INT PRIMARY KEY, MARKS INT, RES VARCHAR(10), RANK
INT, FOREIGN KEY (HTNO) REFERENCES REGISTER(REGNO) ON DELETE CASCADE  ON
UPDATE CASCADE)
```

Foregin Key at Column Level

```
HTNO PRIMARY KEY

FOREIGN KEY REFERENCES REGISTER(REGNO) ON DELETE CASCADE , ON UPDATE
CASCADE
```

Createing Self Referenital Table / Reflexive Relation

```
CREATE TABLE EMPLOYEE( ECODE INT PRIMARY KEY, EN VARCHAR(10), MCODE INT
FOREIGN KEY REFERENCES EMPLOYEE(ECODE))
```

Creating One to Many Relation Ships

```
CREATE TABLE EMP( EID INT PRIMARY KEY, ENAME VARCHAR(10), JOB VARCHAR(10),
DNO INT FOREIGN KEY REFERENCES DEPT(DNO) ON DELETE SET NULL, ON UPDATE SET
NULL)

CREATE TABLE DEPT(DNO IN PRIMARY KEY, DN VARCHAR UNIQUE)
```

NORMALIZATION

It is a process of splitting a single table into two or more tables to avoid redundancy and to promote integrity.

It is a process of deciding number of tables required columns in each table, relationships among tables is known as **NORMALIZATION**.

The aim of Normalization is to store the data in simple format.

Advantages:

- Minimizes redundancy
- Reduces Complexity
- Easy Maintanence
- Making tables ready to perform JOINS and SUBQUERIES

A table can be normalized in the following ways:

- 1NF – FIRST NORMAL FORM
- 2NF – SECOND NORMAL FORM
- 3NF – THIRD NORMAL FORM
- BCNF - BOYCE CODD NORMAL FORM
- 4NF – FOURTH NORMAL FORM
- 5NF – FIFTH NORMAL FORM

FIRST NORMAL FORM

Tables are said to be in FIRST NORMAL FORM if they satisfy the following rules.

- Isolate repeating groups to other table by adding Common Column.
- Every Column should be Atomic.

UNNORMALIZED TABLE :

OrdNo	OrdDate	Cid	Cname	Caddr	Cphone	Items
1	12/7/2011	S12	Sunil	S Nagar	111	P10 PENS 100 B12 BOKS 200 E17 ERSR 200
2	11/7/2011	A13	Shiva	R Nagar	222	P10 PENS 100 S13 SCLS 200
3	10/7/2011	S12	Sunil	S Nagar	111	P10 PENS 300 S13 SCLS 200

To satisfy the First Normal Form, for the above unnormalized table, the below tables can be created to normalize the tables.

Customer			
CID(Primary Key)	Cname	Caddr	CPhone

Order 1			
OrderNo(Primary Key)	Ord_Date	Items	CID(Foregin Key)

Order 2					
OrderNo	Ord_Date	ItemNo	ItemName	Qty	CID(Foreign Key)

SECOND NORMAL FORM

- Tables should be in First Normal Form
- All NON Key Columns should be made dependent on whole Key (i.e Pure Primary Key)

Order 3		
OrderNo(Primary Key)	Ord_Date	CID(Foreign Key)

Items			
ItemNo	ItemName	Qty	OrderNo(Foreign Key)

THIRD NORMAL FORM

- Tables should be in 2NF.
- All the Columns of a table should be made dependent on each other

Unnormalized Table:

WID(Primary Key)	**Skills**	**Bonus**
E10	Electrician	3000
M15	Mechanic	-
C3	-	2500

Workers 1

Skills(Primary Key)	Bonus

Workers2	
WID(Primary Key)	Skills (Foreign Key)

SYNONYMS

It is introduced from SQL SERVER 2005.

- It is created for tables, views where from they can be referred with that name.
- It is a DATABASE Object which resides in DATABASE Server and provides its working till the Base Object exists.
- It is a Permanent Alias Name, any modifications are made through **SYNONYM** will reflect on Base object and vice versa.

Note: SQL Server does not allow to create recursive Synonym.

Syntax:

```
CREATE SYNONYM < SYNONYM_NAME > FOR < BASE_OBJECT >
```

Q) Create a Synonym for SALGRADE Table.

```
CREATE SYNONYM SL FOR SALGRADE
```

Q) Display the List of Synonyms in the DATABASE.

```
SELECT NAME FROM SYS.SYNONYMS
```

Q) Display the List of Synonyms and Base Object Name's in the DATABASE.

```
SELECT NAME, BASE_OBJECT_NAME FROM SYS.SYNONYMS
```

Q) Display the details about the Synonym.

```
SP_HELP < SYNONYM_NAME >

SP_HELP SL
```

Q) Drop a SYNONYM.

```
DROP SYNONYM < SYNONYM_NAME >

DROP SYNONYM SL
```

USER DEFINED SCHEMAS

- Schema will work like a folder.
- It is an object which contains set of different objects.
- A DATABASE Object like tables, views, synonyms.... Etc can be made a part of schema.
- When DATABASE Object are made part of schema, to interact with any object schema name should be used as qualifier.

Syntax: `CREATE SCHEMA < SCHEMA_NAME >`

Q) Creating a Schema.

```
CREATE SCHEMA DBSQLTEST
```

Q) Create a Table under specific Schema.

```
CREATE TABLE DBSQLTEST.EMP( ENO INT, ENAME VARCHAR(10))
```

Q) Display the details about the Schema.

```
SP_HELP 'DBSQLTEST.EMP'
```

Q) Display the User Defined Schemas available in the Current DATABASE

```
SELECT NAME FROM SYSOBJECTS WHERE XTYPE = 'U'
```

Q) Retrive the data from the table a part of a schema.

```
SELECT * FROM DBSQLTEST.EMP
```

Q) Create a Synonym for the Table.

```
CREATE SYNONYM EM FOR DBSQLTEST.EMP
```

CREATING INDEX

- Indexing plays a major role when a data is to be searched in bulk records.
- When data is searched in a normal table it uses Sequential Search Technique, which always a time consuming process or it leads to wastage of time.
- When data is specified with Sequential Search Technique, first it arranges the data in ASCENDING Order of that column and then start searching, whenever it finds the value, it stops the searching process.
- Again and again sorting of the data of that column in Ascending order can be overcome by creating Index.
- In Indexing a separate Index structure is created that includes search key value and address of a record(ROWID) and Index always stores data in Ascending order for permanent.
- When search operation is made first it searches for the data in a related index and then from there it will transfer the control to DATABASE.
- In SQL SERVER 2005 a table can be associated with 250 Index's which One is Clustered Index and 249 is Non Clustered Index are created
- In SQL SERVER 2008 a table can be associated with 1000 Index's which One is Clustered Index and 999 is Non Clustered Index are created

****A table can have more than One Index whereas a column can have only one Index.**

CLASSIFIED IN THREE TYPES

- Clustered Index
- Non Clustered Index
- Unique

Syntax:

```
CREATE [UNIQUE] / [CLUSTERED] / [NON CLUSTERED]
    ON [ TABLE_NAME (COLUMN_NAME…..)]
```

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

CLUSTERED INDEX

- It will Alter the physical representation of rows in a table.
- A table can have only One Clustered Index.
- It will always arrange the data of a table in Ascending order
- Data pages and Index pages will be stored at one level.
- Index should be created for a column where more than one search value is available.

```
CREATE CLUSTERED INDEX INDX1 ON EMP( EMPNO )
```

NON CLUSTERED INDEX

- This index will not Alter the physical requirement of rows in a table.
- A table can have 249 Non Clustered Index's.
- Data is not arranged in Order.
- It is a default index created.
- Data pages & index pages are stored at different level.

```
CREATE NONCLUSTERED INDEX INDX2 ON EMP(ENAME)

CREATE NONCLUSTERED INDEX INDX3 ON EMP(JOB)
```

UNIQUE INDEX

- This index can be created only on those columns which contains unique data.
- This index is automatically created when unique constraint is created on a column.

```
CREATE UNIQUE INDEX INDX4 ON DEPT(DEPTNO)
```

****SQL SERVER creates an Index automatically for two Constraints Primary Key & Unique key.**

TO REBUILD AN INDEX

When a page split occurs for the block which is stored at data page, to provide fragmentation of that page to compress the memory at that data page, index can be rebuild.

Syntax: `ALTER INDEX < INDEX_NAME > ON OBJECT_NAME REBUILD`

Example: `ALTER INDEX INDX1 ON EMP REBUILD`

TO DROP THE INDEX

```
DROP INDEX < TABLE_NAME >.< INDEX_NAME >
DROP INDEX EMP.INDX1
```

TO VIEW THE INFORMATION: `SP_HELPINDEX < TABLE_NAME >`

CREATING VIEWS

- View is a window of virtual table which will allow related users to view and modify related data.

- Any modifications are made through a view will reflect on base table and vice versa , which leads to data consistency
- A view is a database object which resides in database server and stores only select query in compiled format hence it is called stored query.

ADVANTAGES

- Provides Security.
- Improves the performance of a query.
- Network traffic gets reduced.
- Shortens SQL queries.
- Supports to perform DML operations on related data.
- Support to generate summarized reports fastly.

Difference between TABLES and VIEWS

Tables	Views
• Contains Data	• Contains no data.
• Tables are limited.	• Views are Unlimited.
• Independent object	• Dependent Object.
• Supports to DML operations on any data	• It supports to only related data.
• It can be associated with any Trigger	• It Can be associated with only INSTEAD OF TRIGGER

TYPES OF VIEWS

- Simple View
- Check Option View
- Complex View
- Indexed View

****Simple View, Check Option View & Complex View are called as Standard Views.**

SIMPLE VIEW:

- A view which is created on a single table refers to single view and it should not contain group functions, group by clause, distinct operator, joins and mathematical expressions.
- When a simple view is created it should contain all mandatory columns of base table.

Syntax:

```
CREATE VIEW < VIEW_NAME >
[ WITH ENCRYPTION ]/ [ WITH SCHEMA BINDING ]
AS
SELECT QUERY
[ WITH CHECK OPTION ]
```

- In order to store Order by clause through SELECT query into a view, it is must to enclosed TOP keyword.
- When DDL operations are performed on base object they do not get reflected in view, to update the view it should get refreshed, using a predefined procedure.

To Refresh the View

Syntax:

```
SP_REFRESHVIEW < VIEW_NAME >
```

Example:

```
CREATE VIEW V1
AS
SELECT  * FROM EMP

SP_HELP V1 – to get the information about the View.

DELETE FROM V1 WHERE DEPTNO = 10
UPDATE FROM V1 WHERE ENAME = NULL
INSERT V1 (EMPNO,ENAME,SAL,DEPTNO) VALUES (1001,'X',5000,10)
```

Example:

```
ALTER TABLE EMP ADD EXP INT

SP_REFRESHVIEW V1

ALTER TABLE EMP DROP COLUMN HIREDATE

SELECT * FROM V1 -- it would be an error as column is missing.
SP_REFRESHVIEW V1
```

Example:

```
CREATE VIEW V2
AS
SELECT TOP(5) EMPNO, ENAME, SAL, DEPTNO, FROM EMP ORDER BY SAL
```

**** For ORDER BY Clause TOP keyword is must.**

```
CREATE VIEW V3
AS
SELECT EMPNO, ENAME, DEPTNO FROM EMP WHERE DEPTNO = 10

INSERT V3 VALUES()
```

Check Option View: When this option is associated with a view then it will check the condition specified at where clause before data is inserted or updated into base object.

```
CREATE VIEW V4
AS
SELECT EMPNO, ENAME FROM EMP WHERE DEPTNO = 10 WITH CHECK OPTION
```

Complex View: A view is said to be complex view when it is associated with any of the following

- Group By Clause
- Group Functions
- Distinct Operator
- Joins
- Mathematical Expressions
- By Default Complex View is not updatable view(i.e. read only)

```
CREATE VIEW V5
AS
```

```
SELECT DEPTNO, SUM(SAL) TOTSAL FROM EMP GROUP BY DEPTNO

CREATE VIEW V6
AS
SELECT DISTINCT DEPTNO FROM EMP

CREATE VIEW V7
AS
SELECT ENAME, JOB, SAL, SAL*12 FROM EMP

CREATE VIEW V8
AS
SELECT EMPNO, ENAME, JOB, DNAME, LOC FROM EMP, DEPTNO
 WHERE EMP.DEPTNO = DEPT.DEPTNO
```

To display the Query stored in view SP_HELPTEXT < VIEW_NAME >
To display the metadata of a view SP_HELP < VIEW_NAME >
To display list of views that are dependent on a table
SP_DEPENDS < TABLE_NAME > / < VIEW_NAME >
List of columns, table names that are enclosed in view through SELECT Query

Encrypted View:

- When a view is encrypted it does not support to display the text which is stored in a view.
- SQL Server does not support decryption of view.
- To create this view, it should be associated by with encryption
- The query which is stored in encrypted view can be altered

```
CREATE VIEW V9 WITH ENCRYPTION
AS
SELECT * FROM EMP
```

Schema Binding View: Rules to be followed for creating schema binding view.

- Schema binding view will not support to store select query with * Selection operator.
- It is must that table name specified in select query should be preceded by schema name
- The advantage of creating schema binding views is that it will restrict the dropping of the base objects

```
CREATE VIEW V10 WITH SCHEMA BINDING
AS
SELECT * FROM EMP

DROP TABLE DEPT --- ERROR
```

Indexed View:

- These views will provide the improved performance than standard views.
- These views when gets created should be attached with schema binding and there should be an existing of clustered index.
- These views will provide their importance specially when a query contains aggregate functions
- If a query contains aggregate function it includes following steps for the execution of a query.

```
SELECT DEPT NO, COUNT (*) FROM EMP GROUP BY DEPTNO
```

1) Checks the syntax
2) Compiles the query
3) Execution plan is generated
4) Query gets executed
    i. Takes the rows of database into buffer
    ii. Sorts the data
    iii. Creates a group on each deptno
    iv. Count function will visit to each group to count the number of employees
    v. Displays the output
    vi. All these steps are avoided by creating indexed views.

COMMON TABLE EXPRESSIONS

- It is often called as temporary view.
- It will create an object to store query at session level i.e. it provides its usage at all the queries that are connected to Common Table Expressions.
- Table alias column aliases specified in Common Table Expressions should be used at immediate query which works on that table.
- It does not allow using original columns and original table name.
- Common Table Expressions will support to write a query in 2 ways
    - Non recursive
    - Recursive

At non recursive query, Common Table Expression will contain a select query on some object and it gets referred by the other name through a query which provides its execution

Syntax:

```
WITH TABLE_ALIAS (COLUMN_ALIAS)
AS
( SELECT QUERY)  [ SELECT / INSERT / UPDATE / DELETE ]
```

If non recursive query is enclosed with multiple tables, mathematical expressions, distinct operator etc is restricted with Data Manipulation Language Operations

```
WITH E (ENO, EM, PAY, DN)
AS
(SELECT EMPNO, ENAME, SAL, DEPTNO FROM EMP)
 SELECT ENO, EM, PAY, DN FROM E
  WHERE PAY = (SELECT MAX (PAY) FROM E)
  UNION ALL
 SELECT * FROM E

SELECT E (EMPNO, EN, PAY, DNO)
AS
(SELECT EMPNO, ENAME, SAL, DEPTNO FROM EMP)
 INSERT E VALUES (1001,'X', 1000, 10)

SELECT E (EMPNO, EN, PAY, DNO)
AS
(SELECT EMPNO, ENAME, SAL, DEPTNO FROM EMP)
 DELETE FROM E WHERE DN = 10

SELECT E (EMPNO, EN, PAY, DNO)
```

```
AS
(SELECT EMPNO, ENAME, SAL, DEPTNO, LOC FROM EMP, DEPT
  WHERE EMP.DEPTNO = DEPT.DEPTNO)
```

Recursive Query: Common Table Expression which is created for the storage of select query and the query which is used as a statement of execution will provide with temporary view name

```
WITH V1(I)
AS
(SELECT I = 1
UNION ALL
SELECT I = I + 1 FROM V1 WHERE I <=10)
SELECT I FROM V1
```

SECURITY

SQL SERVER 2005 provides a connection to a database server in 2 ways;

Windows Authentication: At this authentication a user connects to database server using windows accounts i.e. users existing at windows operating system. It is called trusted connection.

SQL SERVER Authentication: At this authentication a user connects to database server using the SQL SERVER accounts. By default SQL SERVER provides a login called SA, for which blank password can't be provided in SQL SERVER 2005, whereas in previous versions blank password for SA can be created.

CREATING A NEW LOGIN

To create a new login, it is essential that user should enter into a database which has got system administration privileges.

Syntax:

```
CREATE LOGIN <LOGIN_NAME> WITH PASSWORD='PASSWORD'
CHECKEXPIRATION = OFF, CHECKPOLICY = OFF
```

CHANGING THE PASSWORD:

To change the password of existing login, the following command is used in that database which has got Administration privileges.

Syntax: `ALTER LOGIN <LOGIN_NAME> WITH PASSWORD='PASSWORD'`

DROPING A LOGIN:

To drop a login user should be in a database which has got administration privileges.

Syntax: `DROP LOGIN <LOGIN_NAME>`

To create a login:

```
USE MASTER
CREATE LOGIN TESTLOGIN WITH PASSWORD = N 'TEST123'
DEFAULT_DATABASE = [TESTDB]
CHECKEXPIRATION = OFF, CHECKPOLICY = OFF
```

Granting server roles to login

```
EXEC MASTER.SP_ADDSRVRROLEMEMBER
@LOGINNAME = 'TESTLOGIN', @ROLENAME= 'SYSADMIN'
```

To Create a User

```
CREATE USER KUMAR FOR LOGIN SHIVA
```

To grant database privileges to user

```
USE TESTDB
EXEC SP_ADDROLEMEMBER 'DB_DATAREADER','KUMAR'
EXEC SP_ADDROLEMEMBER 'DB_DATAWRITER','KUMAR'
EXEC SP_ADDROLEMEMBER 'DB_DATADDLADMIN','KUMAR'
EXEC SP_ADDROLEMEMBER 'DB_OWNER','KUMAR'
```

To Change the password:

```
USE MASTER
ALTER LOGIN TESTLOGIN WITH PASSWORD = 'SAMPLE123'
```

To Drop the Login:

```
USE MASTER
DROP LOGIN TESTLOGIN
```

CREATING A LOGIN WITH GUI:

From view menu click on Object explorer--->Databases-->Security--->Right click on Logins--->click on new login

Login Name: Infy
Select SQL SERVER Authentication
Password: test123
Confirm Password: test123
Deselect Enforce Password Policy
Default Database: Batch2
From SELECT A PAGE window, select User Mapping, and Select the database, where in that user it creates a user with the same login name.
Click OK.

Changing the password and properties of Login:

From View menu click on object explorer--->Databases---->Security--->Logins--->Select the login and right click properties
Change the necessary options and then click ok.

Data Control Language:

- This Language is used to give rights on Database Objects created by one user to get them access by other users.
- It also supports to cancel those given rights.
- It includes GRANT, REVOKE, DENY Commands.

Grant: This Data Control Language command is used to give rights on database objects created by one user to get them access by other users
When permissions are granted it can be for multiple users, with multiple permissions but objects will be only one.

Syntax

```
GRANT PRIVELLAGES [COLUMN_LIST]/ALL ON OBJECT_NAME TO USER1
[USER2,USER3...] /PUBLIC [WITH GRANT OPTION]
```

Revoke: it is used to cancel the given rights

```
REVOKE PRIVILEGES / ALL ON OBJECT NAME FROM USER1 [USER2]/PUBLIC [CASCADE]
```

Privileges include select INSERT, UPDATE, and DELETE etc
All – All Privileges
Object Name – Tables, Views, Synonyms etc
Public refers to All Users.

With Grant Option: refers to privileges granted to one user the same privileges that user can grant to other users

Cascade under revoke refers to canceling the privileges from the main user to whom the owner gives the privileges and those privileges will be cancelled from other users

Deny: This Data Control Language command is used to deny privileges on objects that are granted / not granted

```
DENY PRIVELLAGES/ALL ON OBJECT NAME TO USER1…... PUBLIC
```

```
USE TESTDB
GRANT SELECT ON EMP TO AHMED
GRANT SELECT, DELETE ON DEPT TO AHMED
GRANT ALL ON PROD1 TO KHAN
GRANT SELECT ON SALGRADE TO AHMED WITH GRANT OPTION
GRANT SELECT ON EMP2 TO PUBLIC

REVOKE SELECT ON EMP FROM AHMED
REVOKE SELECT ON SALGRADE FROM AHMED CASCADE
```

TRANSACT – SQL

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

Difference between SQL & TSQL

- SQL is a **NON PROCEDURAL LANGUAGE** since it deals with **WHAT** data to be extracted.
- T-SQL is a **PROCEDURAL LANGUAGE,** it deals with **WHAT** Data to be extracted and **HOW** it should be displayed.

T-SQL supports to submit a Block to server for execution in two ways.

- Anonymous block – it supports to store a block in SQL File.
- Stored Block – it stores the blocks in DataBase Server.
  - Stored Procedures
  - Stored Functions
  - Stored Triggers

Global Variables

- These variables are comes with software (SQL Server).
- These variables can't be used to initialize the values manually.
- These variables will stored with data automatically based on the operations performed by SQL Statements.
- These variables are preceded by "@@".

```
SELECT @@VERSION - returns the version of SQL Server
SELECT @@SERVERNAME - returns the name of the server
SELECT @@SERVICENAME - returns name of the service
SELECT @@ROWCOUNT - returns the number of rows effected by last SQL statment(default is 1)
SELECT @@ERROR - return "0" if there is no error in last executed SQL statment or returns error number.
```

To display the List of ERROR messages

```
SELECT * FROM SYS.MESSAGES
```

LOCAL VARIABLES (SCALAR VARIABLES or BUFFER VARIABLES)

- These variables are created by User.
- These variables are provided memory for temporary; hence they are called as BUFFER variable.
- These variables will allow storing only one value; hence it is called as SCALAR Variable. If more than one value stored it gives priority to last value.
- These variables should be declared before they are used.
- These variables preceded by "@".

Syntax:

```
DECLARE @VARIABLE_NAME DATATYPE [( SIZE )]
```

**** In SQL Server 2008 declaration & Initialization can be carried out simultaneously.**

```
DECLARE @VARIABLE_NAME DATATYPE [( SIZE )] [ =VALUE ]
```

SET: This command is used to set a value for a variable.

```
SET @VARIABLE = VALUE/EXPRESSION/FUNCTION
```

PRINT: It is an Output Statement of T-SQL which performs two tasks Display Messages & Display Memory Values.

```
PRINT ' MESSAGE ' / @VARIABLE
```

Q) Write a program to find the addition of two values.

```
DECLARE @A INT, @B INT, @C INT
  SET @A = 70
  SET @B = 80
  SET @C = @A + @B
PRINT ' SUM OF TWO VALUES '+ CAST(@C AS VARCHAR(4))
```

Output: "SUM OF TWO VALUES 150"

Q) Write a program to find Division of two values.

```
DECLARE @A INT, @B INT, @C FLOAT
  SET @A = 10
  SET @B = 20
  SET @C = @A / @B
PRINT 'QUOTIENT FOR TWO VALUES IS '+ CAST(@C AS VARCHAR(5))
```

Output: "QUOTIENT FOR TWO VALUES IS 0"

```
DECLARE @A INT, @B INT, @C FLOAT
  SET @A = 20
  SET @B = 30
  SET @C = @A / (CAST(@B AS FLOAT))
PRINT 'QUOTIENT FOR TWO VALUES IS '+ CAST(@C AS VARCHAR(15))
```

Output: "QUOTIENT FOR TWO VALUES IS 0.66667"

IF CONDITIONAL STATEMENTS

It is used to compare the data provided with the result on Boolean expressions i.e. True or False.

- Simple IF
- IF Else IF
- Nested IF

Simple IF Condition

- This conditional statement is used to compare the data and allow to write a single true or false statement or block.
- If Multiple statements are to be executed on the basis of conditions then they should be enclosed in a BEGIN and END Block

Syntax:

```
IF  < CONDITION >
    < TRUE STATEMENT >
ELSE
    < FALSE STATEMENT >
```

Or

```
IF < CONDITION >
   BEGIN
     < STATEMENTS >
   END
ELSE
   BEGIN
      < STATEMENTS >
   END
```

Q) Write a program to check whether a given number has perfect square or not.

```
DECLARE @N INT, @S INT
  SET @N = 16
  SET @S = SQRT(@N)
IF @S * @S = @N
   PRINT 'YES'
ELSE
   PRINT 'NO'
```

Q) Write a program to check whether a given number is ODD or EVEN without using MOD operator.

```
DECLARE @N INT, @R INT
  SET @N = 7
  SET @R = @N / 2
IF @R * 2 = @N
  PRINT 'IT IS EVEN NUMBER'
ELSE
  PRINT 'IT IS ODD NUMBER'
```

Q) Write a program to check whether a given String is Palindrome or not.

```
DECLARE @S VARCHAR(12),@R VARCHAR(12)
  SET @S = 'RADAR'
  SET @R = REVERSE(@S)
   IF @S = @R
    PRINT ' IT IS PALINDROME '
   ELSE
    PRINT ' SORRY '
```

GOTO STATEMENT:

- It is called branching statement, since it supports to transfer the control from one part of a program to the other part of a program.
- It works with Label assignment and Label Definition.

Label Assignment: `GOTO < Label_Name >`

Label Definition:
Label_Name
< STATEMENTS >

****Label Defined with no label assignment is valid.**
****Label Assigned but no label definition is invalid.**

Example:

```
PRINT 'WELCOME'
GOTO X
Y:
PRINT 'THREE'
PRINT 'NINE'
GOTO Z
X:
PRINT 'SEVEN'
GOTO Y
Z:
PRINT 'THANX'
```

LOOPING CONSTRUCTS

LOOP: Execution of one or more statements repeatedly until a condition is satisfied. SQL Server supports to work with the following Looping constructs.

WHILE LOOPING CONSTRUCT

```
WHILE < CONDITION >
  BEGIN
    < STATEMENTS >
  END
```

Q) Write a program to display first Five Natural Numbers.

```
DECLARE @I INT
SET @I = 1
WHILE @I<=5
BEGIN
  PRINT @I
  SET @I = @I+1
END
```

Q) Write a program to display N Odd Numbers.

```
DECLARE @N INT,@I INT
SET @N = 10
SET @I = 1
WHILE @I<=@N*2
BEGIN
  PRINT @I
  SET @I = @I+2
END
```

Q) Write a program to display reverse of a string without using reverse function.

```
DECLARE @S VARCHAR(10), @R VARCHAR(10),@L INT
  SET @S = 'COMPUTER'
  SET @R = SUBSTRING(@S,LEN(@S),1)
  SET @L = LEN(@S)-1
  WHILE @L>=1
    BEGIN
      SET @R = @R + SUBSTRING(@S,@L,1)
      SET @L = @L-1
    END
    PRINT 'REVERSE IS ' +@R
```

Q) Write a program to count number of vowels and consonants in a given string.

```
DECLARE @S VARCHAR(10),@I INT,@V INT, @C INT
SET @S = 'COMPUTER'
SET @V = 0
SET @C = 0
SET @I = 1
WHILE @I < =LEN(@S)
  BEGIN
    IF SUBSTRING(@S,@I,1) IN('A','E','I','O','U')
      SET @V = @V + 1
```

```
    ELSE
      SET @C = @C + 1
      SET @I = @I + 1
 END
    PRINT 'VOWELS IN THE STRING '+@S+' ARE '+CAST(@V AS VARCHAR(5))
    PRINT 'CONSONANTS IN THE STRING '+@S+' ARE '+CAST(@C AS VARCHAR(5))
```

Q) Write a program to display the given string into substrings eliminating special characters.

```
DECLARE @W VARCHAR(50),@W1 VARCHAR(50)
DECLARE @I INT
SET @W='XYZ;ABC;PQR;AAA;BBB'
SET @I=CHARINDEX(';',@W)
WHILE @I>0
BEGIN
  SET @W1 = LEFT(@W,@I-1)
    IF LEN(@W1)!= 0
      PRINT @W1
        SET @W = SUBSTRING(@W,@I+1,LEN(@W))
        SET @I = CHARINDEX(';',@W)
      IF @I = 0
        PRINT @W
END
```

BREAK & CONTINUE:

```
DECLARE @X INT
SET @X=1
WHILE @X<=10
BEGIN
PRINT @X
    IF @X%3=0
      BREAK
SET @X=@X+1
END
```

```
DECLARE @X INT
SET @X=1
WHILE @X<=10
BEGIN
    IF @X%3=0
    BEGIN
      SET @X=@X+1
      CONTINUE
    END
    PRINT @X
    SET @X=@X+1
END
```

Q) Write a program to check whether a given number is perfect or not.

Perfect Number: Sum of factors of a given number excluding itself should be equal to resultant number.

N = 4, FACTORS: 1, 2 SUM = 1+2 = 3 {NOT PERFECT}
N = 6, FACTORS: 1, 2, 3 SUM = 1+2+3 = 6 {PERFECT NUMBER}
N=28, FACTORS: 1, 2, 4,7,14 SUM = 1+2+4+7+14 = 28 {PERFECT NUMBER}

```
DECLARE @N INT, @S INT, @I INT
SET @N = 4
SET @S = 0
SET @I = 1
WHILE @I <= @N/2
BEGIN
IF @N%@I = 0
      SET @S = @S + @I
      SET @I = @I+1
END
```

```
IF @S = @N
  PRINT ' IT IS A PERFECT NUMBER '
ELSE
  PRINT 'IT IS NOT A PERFECT NUMBER'
```

EXCEPTION HANDLING:

Errors are classified into 3 types:

Complex Time Errors: These errors are often called as Syntax Errors. These errors are occurred when the statements are written beyond the rule.

Logical Errors: Required output and generated output, when does not get match such type of errors are called Logical Errors.

Runtime Errors: These errors will occur during the execution of a program. When these errors occur in a program which is in execution in between gets terminated. In order to have a smooth execution of program exception handling is used.

- In SQL Server 2000 Exception handling was implemented using global variable @@ERROR
- In order to raise the error with respect to user defined error message, a predefined statement called RAISERROR is used.

Syntax:

```
RAISERROR (ARG1,ARG2,ARG3)
```

ARG1 is Error Message
ARG2 Is Severity, which is an INT type data. It can be in a range of 0 to 25.
0 to 9 - displays error message as user defined message (10 is similar to 0)
11 to 18 - displays user defined error message as predefined error message
19 to 25 - it can be associated with those users who have got sysadmin privileges
ARG3 is State, which is also an INT type data. It can be in a range of 1 to 127.
It provides the identification of error when the same error occurred at different areas of a program.

```
DECLARE @A INT,@B INT,@C INT
PRINT 'WELCOME'
SET @A=5
SET @B=0
SET @C=@A/@B
IF @@ERROR <> 0
  RAISERROR('CANT DIVIDE THE NUMBER BY 0',19,1)
    PRINT 'QUOTIENT IS.....'+CAST(@C AS VARCHAR(10))
    PRINT 'THANX'
```

**** The above program is as per SQL Server 2000**

Adding User Defined Error Messages into System Table

```
SP_ADDMESSAGE MESSAGE_ID, SERVERITY_LEVEL,'MESSAGE_TEXT'
SP_ADDMESSAGE 50000,5,'VALUE SHOULD BE BETWEEN 10 AND 15'

DECLARE @A INT
SET @A = 50
IF @A BETWEEN 100 AND 200
```

```
  RAISERROR (50000,15,16)
ELSE
  PRINT @A

SP_DROPMESSAGE MESSAGE_ID
```

STRUCTURED EXCEPTION HANDLING:

In SQL Server 2005 exception handling is carried out using Structured Exception, which supports to write two blocks.

TRY: It is used to monitor all those instructions in which run time errors are expected

CATCH: It is used to catch the thrown exceptions. This block is executed only when exception is raised.

Syntax:

```
BEGIN TRY
      STATEMENTS
END TRY
BEGIN CATCH
      STATEMENTS
END CATCH
```

Note:

- Nested Try blocks are valid.
- Multiple Try and Catch blocks are valid.
- Following are the functions are used to display information about errors.
  - ERROR_SEVERITY()
  - ERROR_STATE()
  - ERROR_LINE()
  - ERROR_MESSAGE()
  - ERROR_NUMBER()

Q) Write a program to find the division of two values and handle the necessary exception using TRY & CATCH Blocks.

```
DECLARE @A INT,@B INT,@C INT
SET @A = 5
SET @B = 0
BEGIN TRY
   SET @C = @A/@B
   PRINT 'QUOTIENT IS.....'+CAST(@C AS VARCHAR(5))
END TRY
BEGIN CATCH
   PRINT 'CANT DIVIDE THE NUMBER BY 0'
   PRINT ERROR_NUMBER()
   PRINT ERROR_LINE()
   PRINT ERROR_MESSAGE()
   PRINT ERROR_SEVERITY()
   PRINT ERROR_STATE()
END CATCH
PRINT 'THANX'
```

Q) Write a program to find square root of a given number and handle the necessary exceptions.

```
DECLARE @N INT, @S INT
SET @N = 144
```

```
  BEGIN TRY
    SET @S = SQRT(@N)
    PRINT 'SQUARE ROOT IS ' + CAST(@S AS VARCHAR(5))
  END TRY
  BEGIN CATCH
    PRINT 'CANT FIND THE SQUARE ROOT '
  END CATCH
```

Q) Write a program illustrating that runtime error which occurs when the data exceeds the capacity of Data Type.

```
DECLARE @N TINYINT
BEGIN TRY
  SET @N = 200
    PRINT @N
END TRY
BEGIN CATCH
      PRINT 'VALUE EXCEEDS'
END CATCH
```

Q) Write a program to handle that runtime error which occurs when data stored in child table does not exist at parent table.

```
CREATE TABLE REGISTER (REGNO INT CONSTRAINT REGNO_PK PRIMARY KEY)

INSERT INTO REGISTER VALUES(101)
INSERT INTO REGISTER VALUES(102)
INSERT INTO REGISTER VALUES(109)
INSERT INTO REGISTER VALUES(104)

CREATE TABLE RESULTS (HTNO INT CONSTRAINT HTNO_FK FOREIGN KEY REFERENCES
REGISTER(REGNO))

BEGIN TRY
      INSERT INTO RESULTS VALUES(201)
END TRY
BEGIN CATCH
      PRINT 'VALUE DOES NOT EXIST AT PARENT TABLE'
END CATCH
```

MULTIPLE TRY CATCH BLOCKS

Q) Write a program to Store the run time errors into a table.

```
CREATE TABLE ERROR_LOG (ERNO INT,ERLI INT,ERMSG VARCHAR(300),ERSEVER INT)

DECLARE @A INT,@B INT,@C INT,@N INT,@S INT
SET @A = 5
SET @B = 0
BEGIN TRY
      SET @C = @A/@B
      PRINT @C
END TRY
BEGIN CATCH
    PRINT 'CANT DIVIDE THE NUMBER BY 0'
```

```
    INSERT ERROR_LOG SELECT ERROR_NUMBER(),ERROR_LINE(),
                                          ERROR_MESSAGE(),ERROR_SEVERITY()
END CATCH
BEGIN TRY
      SET @N = -8
      SET @S = SQRT(@N)
      PRINT @S
END TRY
BEGIN CATCH
    PRINT 'CANT FIND THE SQUARE ROOT OF -VE VALUE'
    INSERT ERROR_LOG SELECT ERROR_NUMBER(),ERROR_LINE(),

      ERROR_MESSAGE(),ERROR_SEVERITY()
END CATCH
```

NESTED TRY CATCH BLOCKS

```
SET NOCOUNT ON
DECLARE @A INT,@B INT,@C INT,@N INT,@S INT
SET @A = 5
SET @B = 0
BEGIN TRY
   SET @C = @A/@B
     PRINT @C
   BEGIN TRY
       SET @N = -8
       SET @S = SQRT(@N)
        PRINT @S
   END TRY
   BEGIN CATCH
      PRINT 'CANT FIND THE SQUARE ROOT OF -VE VALUE'
      INSERT ERROR_LOG SELECT ERROR_NUMBER(),ERROR_LINE(),
                                    ERROR_MESSAGE(),ERROR_SEVERITY()
   END CATCH
  PRINT 'WE ARE OUT OF INNER TRY AND CATCH BLOCK'
END TRY
BEGIN CATCH
    PRINT 'CANT DIVIDE THE NUMBER BY 0'
    INSERT ERROR_LOG SELECT ERROR_NUMBER(),ERROR_LINE(),
                                    ERROR_MESSAGE(),ERROR_SEVERITY()
END CATCH
```

Working with EXEC (Execute): This Command is used to execute the queries that are stored in variables.

Syntax: `EXEC (VARIABLE)`

****The Query stored in a variable can be passed with arguments through variables.**

```
DECLARE @S VARCHAR(200)
SET @S = 'SELECT * FROM EMP'

EXEC (@S)

DECLARE @S VARCHAR(200)
SET @S = 'SELECT * FROM EMP WHERE DEPTNO =10'
```

```
EXEC (@S)

DECLARE @S VARCHAR(200),@DNO INT
SET @DNO = INT
SET @S = 'SELECT * FROM EMP WHERE DEPTNO ='+CAST(@DNO AS VARCHAR(3))

EXEC (@S)

DECLARE @S VARCHAR(200)
SET @S = 'INSERT EMP(EMPNO,ENAME) VALUES(1001,''XYZ'')'

EXEC (@S)
```

SUB PROGRAMS:

- It is a process of splitting a large application program into small modules or blocks.
- SQL Server supports to write a sub program based on the following concepts
  Stored Procedures, Stored Functions, Stored Triggers

Advantages of Sub Programs:

- Provides security
- It improves performance
- Reduces Network Traffic
- Readability gets increases
- Code Reusability
- Error Detection and modification is quite easy
- Extensibility - It will allow a user to increase or decrease the code

STORED PROCEDURES:

SQL Server Supports to work with the following types of procedures

- Predefined Procedures (`SP_HELP, SP_RENAME…….`)
- User Defined Stored Procedures
- Extended Stored Procedures (`XP_CMDSHELL,XP_READMAIL,XP_SENDMAIL`)

USER DEFINED STORED PROCEDURES:

- It is a stored block Data Base Object which resides in Data Base Server and contains SQL Statements and Procedural Features.
- A procedure in SQL Server 2005 can be made to return value to main program using RETURN Key Word.
- It can be created with and without arguments.
- Data can be sent to procedures using INPUT Arguments, after processing that input few values can be taken back to the main program using OUTPUT Arguments.
- A procedure can be set with 1024 INPUT & OUTPUT Arguments.
- When a Procedure is set with OUTPUT arguments then it is essential that at sending arguments also OUPUT should be specified.
- OUTPUT Arguments at a procedure which gets data the same will be reflected in main program.
- Procedure Set with INPUT Arguments will work for “call by value” and OUTPUT arguments will work for “call by reference”

Syntax:

```
CREATE PROC / PROCEDURE < PROCEDURE_NAME > [(LIST_OF_ARGUMENTS)]
[ WITH ENCRYPTION ]
AS
BEGIN
      [ DECLARATION BLOCKS ]
      [ EXECUTION BLOCKS ]
      [ EXCEPTION HANDLERS]
END
```

A Procedure can be executed in two ways:

- Using EXEC Command
- Using Anonymous Block / Main Program

Syntax: `EXEC < PROCEDURE_NAME > [(LIST_OF_ARGUMENTS)]`

****A Procedure can be called for execution in the other Procedure up to 32 Levels.**

Q) Create a Procedure called ADDONE that takes two values as input and finds sum of the values.

```
CREATE PROCEDURE ADDONE( @A INT, @B INT)
AS
BEGIN
  DECLARE @C INT
  SET @C = @A + @B
    PRINT 'SUM OF TWO VALUES IS ' + CAST(@C AS VARCHAR(10))
END
EXEC ADDONE 20,50 -- Output: " SUM OF TWO VALUES IS 70 "
```

Q) Create a Procedure called DIV that takes two values as input and finds the quotient, handle the necessary exceptions.

```
CREATE PROCEDURE DIV( @A INT, @B INT)
AS
BEGIN
  DECLARE @C FLOAT
    BEGIN TRY
      SET @C = @A/CAST(@B AS FLOAT)
        PRINT ' QUOTIENT OF TWO VALUES IS ' + CAST(@C AS VARCHAR(10))
    END TRY
    BEGIN CATCH
      PRINT ' CANT DIVIDE THE NUMBERS BY ZERO '
    END CATCH
END

EXEC DIV 20,20 Output: "QUOTIENT OF TWO VALUES IS 1"
```

Q) Create a Procedure called ADDTWO that take two values as input and returns the result to main program through output arguments.

```
CREATE PROCEDURE ADDTWO(@A INT, @B INT, @C INT OUTPUT)
AS
BEGIN
  SET @C = @A + @B
END
```

```
DECLARE @X INT, @Y INT,@Z INT
  SET @X = 18
  SET @Y = 20
EXEC ADDTWO @X,@Y,@Z OUTPUT
PRINT 'SUM OF TWO VALUES IS ' +CAST(@Z AS VARCHAR(10))
```
Output: "SUM OF TWO VALUES IS 38"

Q) Create a Procedure called RET1 that will retrive all the records of EMP Table, This Procedure will call RET2 to display all the records of DEPT Table and it calls the other Procedure RET3 that will display all the records of SALGRADE Table.

```
CREATE PROCEDURE RET3
AS
BEGIN
      SELECT * FROM SALGRADE
END

CREATE PROCEDURE RET2
AS
BEGIN
      SELECT * FROM DEPT
      EXEC RET3
END

CREATE PROCEDURE RET1
AS
BEGIN
      SELECT * FROM EMP
      EXEC RET2
END
```

`EXEC RET1` **Output:** EMP,SALGRADE,DEPT all three tables will get displayed.
**** Up to 32 Levels Procedures can be created.**

Q) Create a Procedure called ERET1 that takes EMP No as input and displays ENAME, JOB, SAL and DEPTNO.

```
CREATE PROCEDURE ERET1 (@ENO INT)
AS
BEGIN
      SELECT ENAME,JOB,SAL,DEPTNO FROM EMP
       WHERE EMPNO = @ENO
END
```

```
EXEC ERET1 7369
```

	ENAME	JOB	SAL	DEPTNO
1	SMITH	CLERK	800	20

Second Method

```
CREATE PROCEDURE ERET2 (@ENO INT)
AS
BEGIN
  DECLARE @EN VARCHAR(15), @DES VARCHAR(10), @PAY INT, @DNO INT
   SELECT @EN = ENAME, @DES = JOB, @PAY = SAL, @DNO = DEPTNO FROM EMP
    WHERE EMPNO = @ENO
```

```
     PRINT @EN
     PRINT @DES
     PRINT @PAY
     PRINT @ENO
     PRINT @DNO
END

EXEC ERET2 7369
```

Third Method

```
CREATE PROCEDURE ERET3 (@ENO INT ,@EN VARCHAR(15) OUTPUT,
@DES VARCHAR(10) OUTPUT, @PAY INT OUTPUT , @DNO INT OUTPUT)
AS
BEGIN
      SELECT @EN = ENAME,@DES = JOB, @PAY = SAL, @DNO = DEPTNO FROM EMP
       WHERE EMPNO = @ENO
END

DECLARE @EC INT, @N VARCHAR(10), @J VARCHAR(10),@DC INT, @P INT
      SET @EC = 7788
EXEC ERET3 @EC,@N OUTPUT, @J OUTPUT, @DC OUTPUT,@P OUTPUT
      PRINT @EC
      PRINT @N
      PRINT @J
      PRINT @DC
      PRINT @P
```

Q) Create a Procedure to insert a record in EMP by checking different validations EMPNO, ENAME, SAL and DEPTNO.

```
CREATE PROCEDURE INSROW(@ENO INT,@EN VARCHAR(20),@PAY INT, @DNO INT)
AS
BEGIN
SET NOCOUNT ON
  DECLARE @I INT, @J INT, @K INT
    SELECT @I =COUNT(*) FROM EMP WHERE EMPNO =@ENO
    SELECT @J =COUNT(*) FROM DEPT WHERE DEPTNO =@DNO
    SELECT @K = MIN(SAL) FROM EMP
    IF @I > 0
      PRINT 'EMPLOYEE DETAILS ALREADY EXISTS'
    ELSE
     IF @J = 0
     PRINT 'DEPARTMENT IS NOT AVAILABLE'
     ELSE
      IF @PAY < @K
       PRINT 'INVALID SALARY'
    ELSE
      BEGIN
        BEGIN TRAN
        INSERT EMP(EMPNO,ENAME,SAL,DEPTNO) VALUES(@ENO,@EN,@PAY,@DNO)
        COMMIT
        PRINT 'DETAILS INSERTED SUCCESSFULLY'
      END
END

EXEC INSROW 7369,'test',5000,50  --Employee Details Already Exists
```

```
EXEC INSROW 73690,'test',5000,50 --Department Is Not Available
EXEC INSROW 73690,'test',5000,10 --Details Inserted Successfully
```

Q) Create a procedure to update the salary of employee if it is valid increment of more than 500.

```
CREATE PROCEDURE UPDROW(@ENO INT,@INCR INT)
AS
BEGIN
  SET NOCOUNT ON
  DECLARE @I INT
  SELECT @I = COUNT(*) FROM EMP WHERE EMPNO=@ENO
  IF @I = 0
    PRINT 'EMPLOYEE DOES NOT EXIST'
  ELSE
  IF @INCR < 500
    PRINT 'INVALID INCREMENT'
  ELSE
  BEGIN
      BEGIN TRAN
      UPDATE EMP SET SAL=SAL+@INCR WHERE EMPNO=@ENO
    COMMIT
    PRINT 'ROW UPDATED SUCCESSFULLY'
  END
END

EXEC UPDROW 7788,200    -- Invalid Increment
EXEC UPDROW 2001,1200   -- Employee Does Not Exist
EXEC UPDROW 7788,1000   -- Row Updated Successfully
```

Q) Create a procedure to delete the rows from a table based on valid deptno.

```
CREATE PROCEDURE DELROW(@DNO INT)
AS
BEGIN
  DECLARE @I INT, @J INT
  SELECT @I = COUNT(*) FROM DEPT WHERE DEPTNO = @DNO
  IF @I = 0
    PRINT ' INVALID DEPARTMENT '
      ELSE
        BEGIN
         SELECT @J = COUNT(*) FROM EMP WHERE DEPTNO = @DNO
         IF @J = 0
           PRINT 'NO EMPLOYEES EXISTS'
         ELSE
         BEGIN
           BEGIN TRAN
           DELETE FROM EMP WHERE DEPTNO =@DNO
           PRINT CAST(@J AS VARCHAR(10))+' ROWS DELETED SUCCESSFULLY'
         END
    END
END
```

Q) Create a Procedure that performs Bank Transactions provided its implementation on two tables.

```
CREATE TABLE BANK (ACNO INT,AHN VARCHAR(12),ADDRESS VARCHAR(15),BAL INT)

INSERT BANK VALUES(1001,'HARI','AMRPT',12340)
```

```
INSERT BANK VALUES(1007,'KIRAN','BHEL',12900)
INSERT BANK VALUES(1002,'RAJ','ECIL',15400)
INSERT BANK VALUES(1009,'KARAN','AMRPT',23800)
INSERT BANK VALUES(1004,'SUNIL','ABIDS',34900)

CREATE TABLE TRANS (ACNO INT,TT VARCHAR(5),AMOUNT INT,DAT SMALLDATETIME)

CREATE PROC BTRAN(@ACC INT,@TRTY VARCHAR(5),@AMT INT)
AS
BEGIN
  SET NOCOUNT ON
  DECLARE @I INT,@B INT
    SELECT @I = COUNT(*) FROM BANK WHERE ACNO=@ACC
    SELECT @B = BAL FROM BANK WHERE ACNO=@ACC
      IF @I = 0
        PRINT 'INVALID ACCOUNT NUMBER'
      ELSE
      IF @TRTY = 'D'
      BEGIN
      BEGIN TRAN
        INSERT TRANS VALUES(@ACC,@TRTY,@AMT,GETDATE())
        UPDATE BANK SET BAL = BAL+@AMT WHERE ACNO=@ACC
        COMMIT
        PRINT 'ROW INSERTED AND UPDATED SUCCESSFULLY'
      END
        ELSE
        BEGIN
          IF @TRTY = 'W'
          BEGIN
            IF @AMT > @B
              PRINT 'OVER DRAFT OF AMOUNT'
            ELSE
              IF @AMT > @B-500
                PRINT 'INVALID TRANS - MIN BAL SHOULD BE 500'
              ELSE
              IF @AMT < 100
                PRINT 'INVALID AMOUNT'
              ELSE
                BEGIN
                  BEGIN TRAN
                    INSERT TRANS VALUES(@ACC,@TRTY,@AMT,GETDATE())
                    UPDATE BANK SET BAL=BAL-@AMT WHERE ACNO=@ACC
                    COMMIT
                    PRINT 'ROW INSERTED AND UPDATED SUCCESSFULLY'
                END
              END
            ELSE
            PRINT 'INVALID TRANSACTION TYPE'
        END
END
```

MODIFYING THE PROCEDURE

```
ALTER PROC/PROCEDURE < PROCEDURE_NAME >([LIST_OF_ARGS])
[WITH ENCRYPTION]
AS
BEGIN
   STATEMENTS
```

```
END
```

DROPING THE PROCEDURE

```
DROP PROC / PROCEDURE < PROCEDURE_NAME >
```

To display all the list of procedures: SP_STORED_PROCEDURES
To display all the names of stored procedures:

```
SELECT NAME FROM SYSOBJECTS WHERE XTYPE='P'
```

To display information about a specified procedure: SP_HELP 'PROCEDURE_NAME'
To display the code of a stored procedure: SP_HELPTEXT 'PROCEDURE_NAME'

DYNAMIC QUERIES

A query is a block is provided with input at the execution of a program, so that dynamically changes can be made to the query, and will generate the different output.

Q) Retrieving the data from any table.

```
CREATE PROC RETTAB(@TN VARCHAR(12))
AS
BEGIN
 EXEC ('SELECT * FROM '+@TN )
END

EXEC RETTAB EMP
EXEC RETTAB DEPT
```

Q) Create a table using procedure with passing table names at runtime:

```
CREATE PROC CRETAB(@TN VARCHAR(12))
AS
BEGIN
      EXEC ('CREATE TABLE '+@TN+'(ENO INT,EN VARCHAR(12))')
END

EXEC CRETAB E1
EXEC CRETAB E2
```

PROCEDURES RETURNING VALUES

SQL server will allow a procedure to return 1 value of only numeric type.

Syntax:

```
CREATE PROC / PROCEDURE PROC_NAME(LIST_OF_ARGS)
[WITH ENCRYPTION]
AS
BEGIN
      STATEMENTS
      RETURN VARIABLE/VALUE
END
```

TO CALL A PROCEDURE FOR EXECUTION

Syntax:

```
EXEC @VARIABLE=PROC_NAME LIST_OF_ARGS
```

Example:

```
CREATE PROCEDURE ATWO(@A INT,@B INT)
AS
BEGIN
  DECLARE @C INT
  SET @C = @A+@B
  RETURN @C
END

DECLARE @X INT,@Y INT,@Z INT
SET @X = 26
SET @Y = 89
EXEC @Z = ATWO @X,@Y
PRINT 'ADDITION IS.......'+CAST(@Z AS VARCHAR(10))
```

STORED FUNCTIONS (USER DEFINED FUNCTIONS)

- Function is defined as "self contained or predefined program which returns max of 1 value".
- Returned value can be of int, text, datetime.

User defined functions are classified into two types:

- Scalar Valued Functions (which returns 1 value)
- Table Valued Functions (which returns multiple rows)

Scalar Valued Functions: These functions are stored as database objects in a database server and can return max of 1 value. It takes the input thru arguments and returns 1 value after processing those arguments.

Syntax:

```
CREATE FUNCTION FUNC_NAME(LIST_OF_ARGS)
RETURNS DATA_TYPE[(SIZE)]
[WITH ENCRYPTION]
AS
BEGIN
   [DECLARATION PART]
   [EXECUTION PART]
   RETURN VAR/VAL
END
```

Calling a Function for Execution: User Defined Functions should be called for execution by using Schema Name preceded to Function Name.

Using SELECT Command -- `SELECT DBO.FUNCTION_NAME (ARG_LIST)`
Using Anonymous Block -- `SET @VAR = DBO.FUNCTION_NAME (ARG_LIST)`

Q) Create a function that takes two values as input and returns Product of two values.

```
CREATE FUNCTION PROD(@A INT,@B INT)
RETURNS INT
AS
```

```
BEGIN
  DECLARE @C INT
  SET @C = @A*@B
  RETURN @C
END

SELECT DBO.PROD(12,12)
```

Using Anonymous Block

```
DECLARE @X INT, @Y INT,@Z INT
SET @X = 5
SET @Y = 8
SET @Z = DBO.PROD(@X,@Y)
  PRINT 'PRODUCT OF 2 NUMBERS IS '+ CAST(@Z AS VARCHAR(5))
```

Q) Create a function called Fact that takes one number as input to a function and returns factorial of a given number.

```
CREATE FUNCTION FACT(@N INT)
RETURNS INT
AS
BEGIN
   DECLARE @F INT
   SET @F = 1
   WHILE @N > 0
   BEGIN
     SET @F = @F * @N
     SET @N = @N - 1
   END
   RETURN @F
END

SELECT DBO.FACT(5)
```

Q) Create a function that takes empno as an argument and returns employee name.

```
CREATE FUNCTION FIND_EMP(@ENO INT)
RETURNS VARCHAR(12)
AS
BEGIN
  DECLARE @EN VARCHAR(12)
  SELECT @EN = ENAME FROM EMP WHERE EMPNO = @ENO
  RETURN @EN
END

SELECT DBO.FIND_EMP(1005)
```

TABLE VALUED FUNCTIONS or INLINE FUNCTIONS:

- These functions will return multiple rows.
- When these functions are created Begin and End blocks should not be mentioned.
- A function should be provided with return type as TABLE, since it is returning multiple rows.

Syntax:

```
CREATE FUNCTION FUNC_NAME(LIST_OF_ARGS)
```

```
    RETURNS TABLE
[ WITH ENCRYPTION ]
  AS
    RETURN ( SELECT QUERY )
```

Calling a Table Valued Functions:

Syntax:

```
SELECT * FROM DBO.FUNCTION_NAME(LIST_OF_ARGS)
```

Q) Create a function called EMPRET that retrieves the rows of those employees who are working in deptno 20.

```
CREATE FUNCTION EMPRET(@DNO INT)
RETURNS TABLE
AS
   RETURN (SELECT * FROM EMP WHERE DEPTNO=@DNO)
SELECT * FROM DBO.EMPRET(20)
```

Viewing the code related to stored functions: `SP_HELPTEXT < FUNCTION_NAME >`

Dropping a function: `DROP FUNCTION < FUNCTION_NAME >`

To display list of user created scalar functions:

```
SELECT NAME FROM SYSOBJECTS WHERE XTYPE='FN'
```

To display the information about the user created function:

```
SP_HELP < FUNCTION_NAME >
```

TRIGGERS

- It is a stored T-SQL program unit in a database as stored block object, which is associated to a specific table.
- Triggers are executed (fired) automatically when invalid operations are performed over a Trigger Associated table.
- Triggers will not take any arguments like procedures and functions.
- Triggers will not be executed manually as of Procedures and functions.
- Triggering statement will be responsible for the execution of triggers.

ADVANTAGES OF TRIGGERS:

- It can audit the transactions
- We can provide high security for the data and database objects.
- We can provide complex business rules that can't be possible with constraints.
- It can have backup of data without the notice of a user.

TYPES OF TRIGGERS:

- FOR/AFTER Triggers
  - DML Triggers
  - DDL Triggers
- INSTEADOF Triggers
- LOG ON Triggers

DDL Triggers and LOGON Triggers are introduced in SQL SERVER 2005 to perform administration related tasks.

Syntax:

```
CREATE TRIGGER < TRIGGER_NAME >
ON TABLE_NAME/VIEW_NAME/DATABASE
FOR / AFTER / INSTEADOF DML/DDL COMMANDS
AS
BEGIN
      [ DECLARATION PART ]
      [ TRIGGER CONSTRAINT ]
      [ TRIGGER ACTION ]
END
```

VIRTUAL TABLES (MAGIC TABLES)

SQL SERVER provides two virtual tables which can be used only in triggers. These tables are provided by TempDB (System Database)

- These tables play major role to have backup of data.
- Tables are identified as INSERTED, DELETED

INSERTED:

- This table will store the same data which is provided in target table (user defined table).
- If a record is inserted into a target table then the same record is available in this magic table.
- Data available in this table can be used to perform operations and can be again stored into the other user

Defined tables

If the record is updated in a target table, new value is stored in this magic table and old value is transferred to DELETED Table.

DELETED:

- This magic table stores the removed rows from a target table.
- It also stores the old value when update operation is performed over a target table.

Note:

- These tables will store the data for temporary.
- These tables can be used individually or both at a time in a single trigger.

Q) Create a trigger TR1 for INSERT, DELETE triggering event where trigger should be fired if the transactions are performed on SUNDAY.

```
CREATE TRIGGER TR1
ON EMP
FOR INSERT,DELETE
AS
BEGIN
  IF DATENAME(DW,GETDATE())='SUNDAY'
  BEGIN
    ROLLBACK
    RAISERROR('CANT INSERT OR DELETE THE DATA ON SUNDAY',15,16)
  END
END
```

Q) Create a trigger TR2 for INSERT, DELETE Triggering event where trigger should be fired if the transaction is performed before 10AM and after 5PM.

```
CREATE TRIGGER TR2
ON EMP
FOR INSERT,DELETE
AS
BEGIN
  IF DATEPART(HH,GETDATE()) NOT BETWEEN 10 AND 17
  BEGIN
    ROLLBACK
    RAISERROR('INVALID TIME',15,16)
    END
END
```

Q) Create a trigger TR3 for UPDATE triggering event where trigger should be fired to store the updated and old data into a separated table.

```
CREATE TABLE EMPB (ENO INT,OSAL INT,NSAL INT,DOT SMALLDATETIME)

CREATE TRIGGER TR3
ON EMP
FOR UPDATE
AS
BEGIN
  IF UPDATE(SAL)
   INSERT EMPB SELECT I.EMPNO,D.SAL,I.SAL,GETDATE()
   FROM INSERTED I,DELETED D
END
```

Or

```
CREATE TRIGGER TR3
ON EMP
FOR UPDATE
AS
BEGIN
SET NOCOUNT ON
  IF UPDATE(SAL)
  INSERT EMPB SELECT A.EMPNO,A.OLD, A.NEW, A.MDATE
  FROM (SELECT I.EMPNO, D.SAL OLD, I.SAL NEW, GETDATE() MDATE FROM
  INSERTED I, DELETED D WHERE I.EMPNO=D.EMPNO)A
END
```

Q) Create a trigger TR4 for DELETE triggering event on DEPT table, where trigger should be fired by deleting the records from EMP table.

```
CREATE TRIGGER TR4
ON DEPT
FOR DELETE
AS
BEGIN
  SET NOCOUNT ON
  DELETE FROM EMP WHERE DEPTNO IN(SELECT DEPTNO FROM DELETED)
  PRINT CAST(@@ROWCOUNT AS VARCHAR(5))+' Rows Are Deleted'
END
```

Q) Create a trigger for DDL triggering event that restricts the dropping and altering of a table in a database.

```
CREATE TRIGGER TR5
```

```
ON DATABASE
FOR DROP_TABLE,ALTER_TABLE
AS
BEGIN
   ROLLBACK
   RAISERROR('CANT ALTER OR DROP THE TABLES',15,16)
END
```

INSTEADOF TRIGGERS:

- These triggers are used to make the modifications into base table thru a complex view.
- By default a complex view is not updatable view (i.e. read only view).
- A complex view consists of joins, mathematical expressions, group by clause, distinct operator.

Q) Create a complex view on EMP table that stores a query for empno, sal and annual salary.

```
CREATE VIEW V1
AS
SELECT EMPNO,SAL M_SAL,SAL*12 A_SAL FROM EMP

CREATE TRIGGER TR6
ON V1
INSTEAD OF INSERT
AS
BEGIN
   INSERT EMP(EMPNO,SAL) SELECT EMPNO,M_SAL FROM INSERTED
END
```

To display list of triggers created in a database:

```
SELECT NAME FROM SYSOBJECTS WHERE XTYPE='TR'
```

VIEWING TRIGGER INFORMATION:

```
SP_HELPTEXT < TRIGGER_NAME >
```

DISABLE / ENABLE THE TRIGGERS:

```
ALTER TABLE < TABLE_NAME > DISABLE/ENABLE TRIGGER < TRIGGER_NAME>
```

TO DISABLE DATABASE OR DML TRIGGERS:

```
DISABLE/ENABLE  TRIGGER < TRIGGER_NAME > [ALL] ON DATABASE /
                                                    OBJECT_NAME
```

DROPING A TRIGGER: `DROP TRIGGER < TRIGGER_NAME > ON < DATABASE_NAME >`

CURSORS

- It creates a temporary memory at database server to store result set of a query.
- It is created at a system database TempDB
- Cursors will create a memory to store multiple rows that are returned by a SELECT query.

Cursors are Created based on the following steps:

- Declaring a Cursor
- Opening a Cursor
- Fetching the data from a Cursor
- Closing a Cursor

- Deallocating a Cursor

DECLARING A CURSOR:

A cursor is declared by its name associated with a standard SELECT query using DECLARE keyword. It stores a query for retrieving all rows of a table or retrieving the rows on conditional basis. It also includes navigation of cursor, type of cursor and type of lock provided on cursor.

Syntax:

```
DECLARE < CURSOR_NAME > /VAR CURSOR
    NAVIGATION_TYPE FORWARD / SCROLL
    CURSOR_TYPE STATIC / KEYSET / DYNAMIC / FAST FORWARD
    LOCK_TYPE  READONLY / SCROLL LOCK / OPTIMISTIC
    FOR SELECT QUERY
```

****By default a cursor is FORWARD only cursor, which supports to access the cursor sequentially.**
****If navigation type is provided as SCROLL then can access cursor randomly.**

CURSOR TYPES: SQL SERVER will support to create a cursor with any of the following types;

- **STATIC:** When this cursor type is specified it will not make the changes in the cursor when the table is changed in a database.
- **KEYSET:** When this cursor type is specified it will make the changes in a cursor when the table is provided with the changes thru DML operations.
- **DYNAMIC:** When this cursor type is specified it also makes the changes in cursor when the table is changed in a database but performance gets decreases.
- **FAST FORWARD:** When this cursor type is specified it will work similar to KEYSET and provides fast performance.

LOCKS provided on Cursor:

- **READ ONLY (Shared Lock):** When this lock is provided cursor will not accept any changes into it.
- **SCROLL LOCK (Exclusive Lock):** It will allow to make modifications but only by one user.
- **OPTIMISTIC:** It will allow to make modifications but only by multiple users.

OPENING A CURSOR:

A cursor which is created when gets opened; the query which is associated gets executed and makes the data available in a cursor.

Syntax:

```
OPEN < CURSOR_NAME >
```

FETCHING DATA FROM A CURSOR:

Data from a cursor gets fetched in different ways into temporary or buffer variables. It supports to fetch only 1 row at a time.

Syntax:

```
FETCH {FIRST/LAST/NEXT/PREV}
        ABSOLUTE =POSITION
        RELATIVE =POSITION
FROM CURSOR_NAME
INTO TEMPORARY VARIABLES
```

CLOSING A CURSOR:

The created cursor when gets closed it does not support to fetch the data from a cursor.
The operations can be performed over that cursor when it gets reopened.

Syntax: `CLOSE < CURSOR_NAME >`

DEALLOCATING A CURSOR:

When the cursor gets deallocated, it removes the complete structure and does not support to reopen the cursor. If operations are to be performed over a cursor again it should be created.

Syntax: `DEALLOCATE < CURSOR_NAME >`

Note:

- Any number of cursors can be created in a single program but with unique names.
- Global variable `@@FETCH_STATUS` plays an important role, where it returns Boolean value i.e. returns 0 if it makes a fetch operation successful over a cursor else returns -1.

Q) Create a cursor which supports to access all the employ records.

```
DECLARE @EN VARCHAR(12),@DES VARCHAR(14),@DNO INT
DECLARE C1 CURSOR FOR SELECT ENAME,JOB,DEPTNO FROM EMP
OPEN C1
FETCH NEXT FROM C1 INTO @EN,@DES,@DNO
WHILE @@FETCH_STATUS<>-1
BEGIN
  PRINT @EN+'    '+@DES+'    '+CAST(@DNO AS VARCHAR(5))
  FETCH NEXT FROM C1 INTO @EN,@DES,@DNO
END
CLOSE C1
DEALLOCATE C1
```

Q) Write a program to create a cursor to store unique department numbers and the other cursor which stores employees of those departments.

```
DECLARE @ENO INT,@EN VARCHAR(12),@PAY INT,@DNO INT
DECLARE C1 CURSOR FOR SELECT DISTINCT DEPTNO FROM EMP
OPEN C1
  FETCH NEXT FROM C1 INTO @DNO
  WHILE @@FETCH_STATUS<>-1
  BEGIN
    PRINT 'DEPARTMENT........'+CAST(@DNO AS VARCHAR(10))
    PRINT 'EMPNO'+'      '+'ENAME'+'      '+'SALARY'
  DECLARE C2 CURSOR FOR SELECT EMPNO,ENAME,SAL FROM EMP WHERE DEPTNO=@DNO
  OPEN C2
  FETCH NEXT FROM C2 INTO @ENO,@EN,@PAY
  WHILE @@FETCH_STATUS<>-1
    BEGIN
      PRINT CAST (@ENO AS VARCHAR(8))+'      '+@EN+'      '+CAST(@PAY AS
VARCHAR(10))
    FETCH NEXT FROM C2 INTO @ENO,@EN,@PAY
     END
  CLOSE C2
  DEALLOCATE C2
    FETCH NEXT FROM C1 INTO @DNO
  END
```

```
CLOSE C1
DEALLOCATE C1
```

Q) Write a program to create SCROLL cursor and access first and last records of EMP table.

```
DECLARE @ENO INT,@EN VARCHAR(12),@DNO INT
DECLARE C1 CURSOR SCROLL FOR SELECT EMPNO,ENAME,DEPTNO FROM EMP
OPEN C1
  FETCH FIRST FROM C1 INTO @ENO,@EN,@DNO
   IF @@FETCH_STATUS = 0
   BEGIN
     PRINT CAST(@ENO AS VARCHAR(10))+'   '+@EN+'   '+CAST(@DNO AS
VARCHAR(12))
   END
   ELSE
   BEGIN
     PRINT 'FETCHING FAILED'
   END
   FETCH LAST FROM C1 INTO @ENO,@EN,@DNO
   IF @@FETCH_STATUS=0
   BEGIN
     PRINT CAST(@ENO AS VARCHAR(10))+'   '+@EN+'   '+CAST(@DNO AS
VARCHAR(12))
   END
   ELSE
   BEGIN
     PRINT 'FETCHING FAILED'
   END
CLOSE C1
DEALLOCATE C1
```

Q) Write a program to display required record from a cursor using absolute position.

```
DECLARE @ENO INT,@EN VARCHAR(12),@DNO INT
DECLARE C1 CURSOR SCROLL FOR SELECT EMPNO,ENAME,DEPTNO FROM EMP
OPEN C1
  FETCH ABSOLUTE 3 FROM C1 INTO @ENO,@EN,@DNO
  FETCH RELATIVE 2 FROM C1 INTO @ENO,@EN,@DNO
      IF @@FETCH_STATUS=0
      BEGIN
      PRINT CAST(@ENO AS VARCHAR(10))+'   '+@EN+'   '+CAST(@DNO AS
VARCHAR(12))
    END
    ELSE
    BEGIN
      PRINT 'FETCHING FAILED'
    END
CLOSE C1
DEALLOCATE C1
```

Note:

- When RELATIVE position is specified with +Ve value, it moves the record pointer from current position to the next record in forward direction.
- If -Ve value is specified it moves the record pointer in backward direction from current position. To the current position relative position is added or subtracted and then takes the control to the next record.

Q) Write a program to delete all tables from a Database.

```
DECLARE C2 CURSOR FOR SELECT NAME FROM SYS.TABLES
DECLARE @TN VARCHAR(12)
OPEN C2
FETCH NEXT FROM C2 INTO @TN
  WHILE @@FETCH_STATUS <>-1
  BEGIN
    EXEC('DROP TABLE '+@TN)
    FETCH NEXT FROM C2 INTO @TN
  END
CLOSE C2
DEALLOCATE C2
```

CLR INTEGRATION

In SQL Server 2005 Database Engine is integrated with CLR (Common Language Runtime Environment) Database objects can be created with T-SQL Code + DOT NET Language Code.

A Database object created with T-SQL + DOT NET Language Code can be called as CLR Database Object

- CLR Stored Procedure
- CLR Stored Functions
- CLR Stored Triggers
- CLR User Defined Types
- CLR User Defined Aggregates

To prepare CLR Database object we have to follow four steps.

- Develop .NET Assembly
- Register the Assembly in SQL Server 2005.
- Convert the Assembly methods into SP's, SF's, ST's etc.
- Invoke the Object through SQL Server.

CLR Function with C#

```
using.system;
using.system.collections.generic;
using.system.text;
using.system.data;
using.system.data.SQLClients;
using.system.data.SQLTypes;
using.microsoft.SQLserver.server;

Public Class CLRClass
{
        [MS SQL Server SQL Function]
  Public Static Int MyCLRMethod (int a, int b)
        {
          return (a+b);
        }
```

```
}
```

Step 2: Register the Assembly in SQL Server.

GUI

Step 1: Select the Database
Step 2: Click on Programmability
Step 3: Click on New Assembly
Step 4: Provide the Path of the Compiled C# DLL File

Query Based

```
      CREATE ASSEMBLY MYCLRASSEMBLY FROM 'C:MYCLR\BIN\DEBUG\MYCLR.DLL'
        WITH PERMISSION_SET = SAFE

      PERMISSION_SET = SAFE / UNSAFE / EXTERNALACCESS

CREATE FUNCTION MYCLRTEST(@I INT, @J INT)
RETURNS INT
AS
EXTERNAL NAME
MYCLRASSEMBLY.CLRCLASS.MYCLRMETHOD

SELECT DBO.MYCLRTEST(27,89)
```

ENABLE/DISABLE CLR Integration:

Step 1: Microsoft SQL Server
Step 2: Configuration Tools
Step 3: SQL Server Surface Area Configuration
Step 4: Surface Area Configuration Features
Step 5: CLR Integration
Step 6: Enable the CLR

Or

```
SP_CONFIGURE 'SHOW_ADVANCED OPTIONS',1

SP_CONFIGURE 'CLR ENABLED',1
   1- ENABLE, 0 - DISABLE
```

XML INTEGRATION

In SQL SERVER 2005 XML integration is introduced, which supports to store and access the data in XML format.

Retrieving the table data in XML Format:
The data in a table which is in the form of rows and columns can be displayed in XML Format using FOR XML clause of SELECT Statement.

Syntax:

```
      SELECT ....................
      FROM TABLE1...........
      FOR XML <RAW / AUTO / PATH>, ROOT (NAME)
```

```
SELECT EMPNO,ENAME,DEPTNO FROM EMP FOR XML RAW
Output:     <row EMPNO="7369" ENAME="SMITH" DEPTNO="20" />
            <row EMPNO="7499" ENAME="ALLEN" DEPTNO="30" />
            <row EMPNO="7521" ENAME="WARD" DEPTNO="30" />

SELECT EMPNO,ENAME,DEPTNO FROM EMP FOR XML AUTO,ROOT('EMPINFO')
Output:
      <EMPINFO>
             <EMP EMPNO="7369" ENAME="SMITH" DEPTNO="20" />
             <EMP EMPNO="7499" ENAME="ALLEN" DEPTNO="30" />
             <EMP EMPNO="73690" ENAME="test" DEPTNO="10" />
      </EMPINFO>

SELECT EMPNO,ENAME,DEPTNO FROM EMP FOR XML PATH,ROOT('EMPINFO')
Output:     <EMPINFO>
              <row>
                <EMPNO>7369</EMPNO>
                <ENAME>SMITH</ENAME>
                <DEPTNO>20</DEPTNO>
              </row>
              <row>
                <EMPNO>7499</EMPNO>
                <ENAME>ALLEN</ENAME>
                <DEPTNO>30</DEPTNO>
              </row>
            </EMPINFO>
```

Storing XML data into table:

- Data which is represented in XML format can be stored into a table in the form of rows and columns using OPENXML ()
- XML data is converted in table format by creating a temporary memory and it is done by using a predefined procedure.

```
EXEC SP_XML_PREPAREDOCUMENT ARG1 OUTPUT, ARG2
```

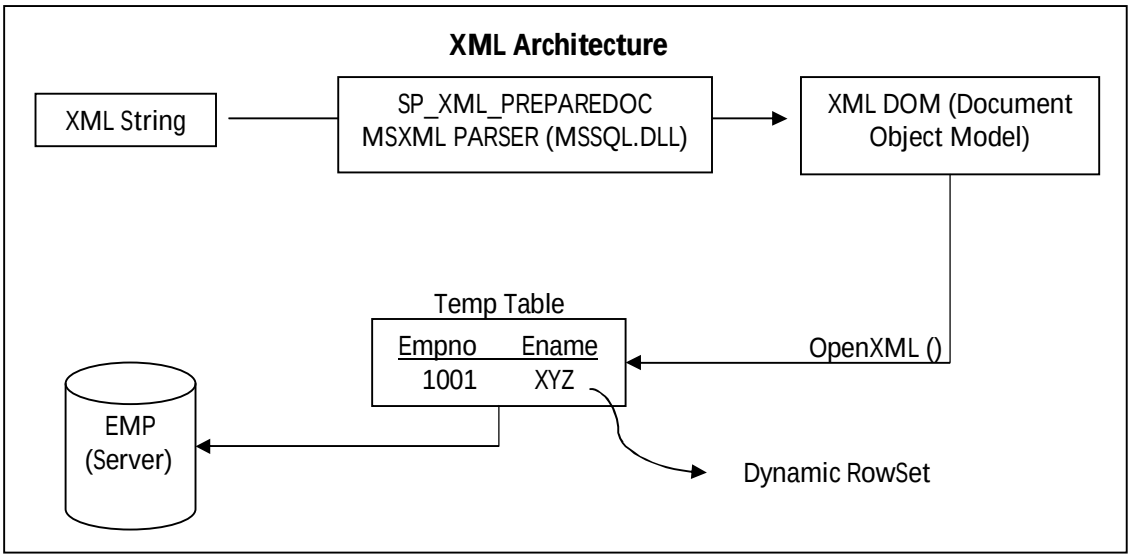
Arg1 is a pointer to that temporary memory table which stores the address of that table. It can be used when that temporary table is to be removed.
Arg2 is a variable which contains the data in XML format.

Use INSERT INTO SELECT query.

To remove that temporary memory table, a predefined procedure is used.

```
EXEC SP_XML_REMOVEDOCUMENT Arg1
```

Arg1 is a pointer variable which contains the address of that temporary table.




```
CREATE TABLE EMPX (ENO INT, EN VARCHAR (12))

DECLARE @XMLDOC VARCHAR (500), @P INT
SET @XMLDOC='<ROOT>
             <EMP ENO="9001" EN="PRASAD" />
             <EMP ENO="9002" EN="SUNIL" />
             <EMP ENO="9010" EN="KIRAN" />
           </ROOT>'
EXEC SP_XML_PREPAREDOCUMENT @P OUTPUT,@XMLDOC
INSERT INTO EMPX SELECT * FROM OPENXML (@P,'/ROOT/EMP',1)
WITH
(ENO INT, EN VARCHAR (12))
EXEC SP_XML_REMOVEDOCUMENT @P
```

Working with XML Data Type:

- XML data type has been introduced in SQL SERVER 2005.
- It supports to insert the data in XML format using INSERT command. XML is case sensitive.
- XML data type provides default memory of 2GB.

```
CREATE TABLE EMPT (ENO INT, INFO XML)

INSERT INTO EMPT VALUES (101,'<EMP>
                        <ENAME>XYZ</ENAME>
                        <SAL>1000</SAL>
                        </EMP>')

INSERT INTO EMPT VALUES (102,'<EMP>
                        <ENAME>ABC</ENAME>
                        <SAL>2000</SAL>
                        </EMP>
                        <EMP>
                        <ENAME>PQR</ENAME>
                        <SAL>3000</SAL>
                        </EMP>')
```

query () : This function will support to retrieve the required attribute information from the XML data stored in a column. This function is case sensitive, which means it should be used with small letters only. This function should be used with column name created with XML data type.

```
SELECT info.query('/EMP/ENAME') FROM EMPT

SELECT info.query('/EMP/SAL') FROM EMPT
```

exist() : This function will return Boolean value i.e. 1 if the specified element exists else returns with 0.

```
SELECT info.exist('/EMP/SAL') FROM EMPT
```

```
SELECT A.ENAME,A.AVER FROM (SELECT ENAME,AVG(MAR) AVER ,ROW_NUMBER() OVER
(ORDER BY AVG(MAR) DESC) AS AA FROM STU
GROUP BY ENAME) A WHERE A.AA = 2
```

TRANSACTION & LOCKS

Transaction is nothing but a logical unit of work, which contains a set of operations. Transaction & Locks increase the complexity of Database operations, by the guarantee valid data and query results in multi user applications. This guarantee is expressed with ACID Properties

Atomic: For a transaction to be ATOMIC all the DML Statements **INSERT, UPDATE, DELETE** must either call COMMIT or ROLLBACK which means a transaction can't be left in half done.

Consistency: Means a user should not never see data changes in the mid of transaction. Their view should return the data as it appears prior to beginning of the transaction or if the transaction is finished. Then they should see the changed data.

Isolation: it is the heart of Multi User Transaction; it means one transaction should not disrupt another transaction.

Durability: Implies that the changes made by the transaction are recorded permanently.
Locking mechanism eliminates concurrency problem and Provides consistency. Locks can be applied at different levels.

LOCK GRANULARITY
The Simplest way to meet the **ACID** requirements would be for a transaction to create a lock that seizes the entire Database affected, not letting anyone touch or see any other data until the transaction completed.

This solution would of course cause a great deal performance problems as other transactions try to access data. The other extreme would be to lock on a single row. A lock can be issued against a Database, Table, Extent, Page or a Row. A Page is usually of 8KB Storage Area on the hard drive. An extent is a group of 8 Pages.

Lock Modes:
SQL Server utilizes several different types of locks & modes. The way in which a lock shares or does not share records.

Exclusive Lock (X): This Lock mode is very straight forward where group of records are taken (Row, Pages, Extent, Table or Database) and are held exclusively by one transaction. When an insert, Updated, Delete Statement runs Exclusive Lock will be issued. No other operation of any kind can use the records hold by X Lock.

Shared Lock: A Shared Lock is applied on records read by a Select Statement. It is designed to allow concurrent read access from other transactions, but none can modify the held records.

Dead Lock: Dead Lock refers to the condition in which one resource is waiting on the action of a second while that second is waiting on first; it neither completes any of the two conditions.

Other Locks: SQL also provides other type of locks, but they will not be used for Lock Optimization. Schema Locks(Sch*) are usually used when a table is being modified such as column being added, Bulk Update Tools are used for Bulk Update Statements , intent Lock(I*) are used internally by SQL to increase performance.

SP_LOCK- It will return the list of information about locks that are available in database.

```
SELECT * FROM SYS.DM_TRAN_LOCKS
```

Example on Shared Lock

```
CREATE TABLE EMPL (ENO INT, EN VARCHAR(10))

INSERT INTO EMPL VALUES (100,'X')
INSERT INTO EMPL VALUES (101,'Y')
INSERT INTO EMPL VALUES (102,'Z')
BEGIN TRAN SHARED
SELECT * FROM EMPL WITH (HOLDLOCK)
SELECT RESOURCE_TYPE, REQUEST_MODE FROM SYS.DM_TRAN_LOCKS
WHERE RESOURCE_TYPE <> 'DATABASE'
WAITFOR DELAY '00:00:10'
```

Example on Exclusive Lock

Session 1:

```
BEGIN TRAN EXCLUSIVE
UPDATE EMP SET ENO = 200
SELECT RESOURCE_TYPE, REQUEST_MODE FROM SYS.DM_TRAN_LOCKS
WHERE RESOURCE_TYPE <> 'DATABASE'
WAITFOR DELAY '00:00:15'
```

Session 2:

```
SELECT * FROM EMPL WITH (NOLOCK)
```

ISOLATION LEVELS:

These are classified in four types:

- READ COMMITTED
- READ UNCOMMITTED
- REPEATABLE READ
- SERIALIZABLE

****By Default a transaction isolation level is "READ COMMITED"**

Syntax:

```
SET TRANSACTION ISOLATION LEVEL < LEVEL_TYPE >
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT * FROM EMPL
```

BULK COPY PROGRAM

- The Bulk Copy Program Utility in SQL Server enables Database administrator to import bulk data into a table or export it from a table into a file.
- It also supports several options that define how data will be imported. Where it will be imported and which data will be loaded.
- BCP Commands will allow a user to Bulk Copy Data "IN" & "OUT" of SQL Server Tables.

Syntax:

BCP < Source.DB > IN / OUT < Destination file > [Options]
IN – Import, OUT – Export

Options are Case Sensitive

-T --- Trusted Connection (Windows Authentication)
-S --- Server Name
-U ---User Name
-P --- Password (SQL Server Authentication)
-c --- Character Format
-r --- Row Terminator (CR-LF) Carriage Return Line Field
-E --- Field Terminator (Tab Default)

Exporting data with BCP Utility one of the simplest operations that can perform with BCP is to bulk copy data out of a SQL Server table into a Text File

Exporting data through Query

BCP "SELECT * FROM TESTDB.DBO.EMP" QUERYOUT C:\emp.csv –c –T -t

Importing Data: To Import the data of a file into a table create at SQL Server, BCP will provide an option called "IN"

BCP testdb.dbo.emp IN C:\emp.csv –c –T –t -E

BACKUP & RECOVERY OF A DATABASE

Backup of a Database: As system is an electronic device, at any point of time it may come across with a problem in hardware or at Operating System or at Database, due to which data available in Database may be lost.
In order to have security for data and Database objects. It is essential that at the End of the Day to go for a Backup. It creates a backup of Database in to Flat Files. Backup Data later can be transferred into other external devices for security reasons.

Backup is done in different ways

- Full Backup
- Differential Backup
- Files & File Groups
- Transaction Logs

Syntax:

```
BACKUP DATABASE <DATABASE_NAME>
TO DISK ='FILEPATH.BAK'
```

Syntax:

```
BACKUP DATABASE <DATABASE_NAME>
TO DISK ='FILEPATH.BAK'
WITH DIFFERENTIAL
```

Restore of Database: Database can be restored in a same system or it can be on different system.

Syntax:

```
RESTORE DATABASE <DATABASE_NAME>
FROM DISK ='FILEPATH.BAK'
```

Note: for Restoring the Query to be executed using master database

EXTENDED STORED PROCEDURES:

A Stored Procedure contains T-SQL Statements and C/C++ Code compiled into DLL file is called Extended Stored Procedure. Every Extended Stored Procedure starts with XP. All Extended Stored Procedures are stored in masterdb.

Extended Stored Procedures are used to perform operations in Operating system and Mail Server.
XP_CMDSHELL (Operating System Command) it is used to communicate with Operating System to execute commands.

Note: to work with this extended procedure it should be enable at configuration tools

```
SP_CONFIGURE 'CMDSHELL',1
RECONFIGURE

XP_CMDSHELL 'DIR C:\'
XP_CMDSHELL 'MD C:\NEW'
```

XP_SENDMAIL, XP_READMAIL are used to interact with mailing servers.

DEPLOYMENT OF DATABASE: The process of copying database objects structures with data from one server to another server is called Deployment of Database.

Method 1:

Copy Backup file (.BAK) to destination system and return it on destination system.

Method 2:

- Copy Data Files and Log Files to destination system.
- Detach the Database on Source System
- SP_DETACH_DB <DATABASE_NAME>
- Copy the Database Files to same location on destination system.
- Attach these files to a Database on destination system

```
CREATE DATABASE < DATABASE_NAME >
ON
FILENAME = 'FILE_PATH'
FOR ATTACH
```

Method 3:

Creating a Batch and Script Files:

- Batch is nothing but set of T-SQL Statements executed under single execution plan.

- Every Batch is followed by GO (which makes a separate execution plan) Creation of any Objects, Setting any property should be presented in different batches.
- Script is nothing but collection of batches saved in a file with .SQL extension.

Advantages:

- Script files are reusable.
- Script files are used to copy database objects from one system to another system.

Generating T-SQL Scripts for existing Database SQL Server provides Script wizard to generate scripts for existing Database and other objects.

Select Database
Click on Tasks
Click on Generate Scripts - Click Next
Select Database Name - Click Next
Check Scripts all objects in Database - Click Next
Provides Scripting options
    Append to File = True (If True Select Existing File)
    Script Database Create = True (If True Create Database statement generated)
    (If False Create Database Statement will not generated)
Script for server version SQL Server 2005
Click Finish

DATA TRANSFORMATION SERVICE or SQL SERVER INTEGRATION SERVICES

- In SQL Server 2000 we have DTS Packages which are based on COM Technology (Component Object Model)
- In SQL Server 2005 it is introduced as SSIS package based on DOTNET Technology.
- To import or export Database objects and data from one source to another data source we can use DTS or SSIS Package.

SSIS Package will support to import or export the database and data between following software's.
    SQL Server ---------- MS Access
    SQL Server ---------- Oracle
    SQL Server ---------- SQL Server
    SQL Server ---------- Text File/XML Files/Excel Files etc

DTS/SSIS communicates with any data source with the help of ODBC or OLEDB or DOTNET data providers.

Note: SSIS Package's will be stored in MSDB Database.

SQL Native Client: Interface can be used to communicate with only SQL Server

Exporting data from SQL Server to Excel Sheet

Select Database
Right Click on Select Export Data - Click Next
Specify Source Data
    Data Source: SQL Native Client

Server Name: Sys-PC
Database: TestDB - Click Next

Specify Destination

Destination: MS EXCEL
File Path: 'C:\Testexcel.xls '
Check First row has column names - Click Next

Select Copy data from one or more tables or views - Click Next
Select Tables/ Views whose data is to be copied into Excel sheet - Click Next
Check Execute Immediately
Click Finish

Importing data from SQL Server to SQL Server

Select Database (Destination Database)
Right Click on Select Import Data - Click Next
Specify Source Data

Data Source: SQL Native Client
Server Name: Sys-PC
Database: TestDB - Click Next

Specify Destination

Destination: SQL Native Client
Server Name: Sys-PC
Database: TestDB1 - Click Next

Select Copy data from one or more tables or views - Click Next
Select Tables/ Views whose data is to be copied into Excel sheet - Click Next
Check Execute Immediately
Click Finish

Browsing SSIS packages

- Object Explore
- Connect to Integration Services
- Select Stored Packages
- MSDB (System Database)
- Select Package and Run Package

SQL SERVER AGENT

- In SQL Server 2005 SQL Server Agent is a service, it supports to create jobs, schedules, alerts and operations.
- It provides automatic execution of Jobs and Tasks associated with different steps automatically based on system date, Day and Time.
- To work with SQL Server Agent it is essential that services should get start.

To start the Service

Start - Programs – SQL Server – Configuration Tools – SQL Server Configuration Manager – Select SQL Server Manager and Click Start

Example:

Create a table

```
CREATE TABLE SSA(ID INT IDENTITY, DATE SMALLDATETIME())
```

To Work with SQL Server Agent:

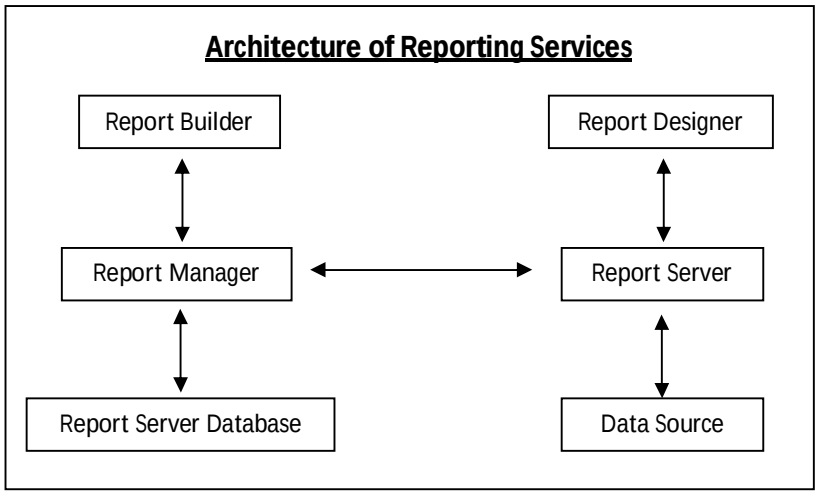
Object Explorer
Click on SQL Server Agent
Step 1:
New Job
Name: INS_REC_JOB
Owner: SA
Category [uncategorized (local)]
Step 2:
Select Steps from select a page window
Click New
Step Name: INS_REC
Type: T-SQL Script
Database: TestDB
Command: `INSERT SSA VALUES (GETDATE())`
Step 3:
Select Schedule from select a page Window
Click New
Name: SCH_INS_REC
Schedule Type: Recurring
Frequency Occurs: Daily
Click Finish

SQL SERVER REPORTING SERVICES

- It is a server based reporting generation software system from Microsoft. It can be used to prepare and deliver a variety of interactive and printed reports.
- It is administrated via a web interface reporting services features a web service interface to support the development of custom reporting applications.
- SSRS Competes with crystal reports and other business intelligence tools and it is included in express, workgroup, standard and enterprise edition of SQL Server as an install option.
- Reporting services was first released in 2004 as an add-on to SQL Server 2000.
- The Second version was released as a part of SQL Server 2005 in November 2005.
- The latest version was released as part of SQL Server 2008 in August 2008.

Reporting services frame work contains the following components.

- Report Server
- Report Manager
- Report Temporary Database



Report Server: It compiles and executes client requests connecting to different data sources and loads the required data in to temporary database.

Report Manager: It will verify security information connections information of client.

Report Temporary Database: To store the reporting data temporary for processing purpose.
Generating Sample Reports:

Starting the Service: Configuration Tools – SQL Server Configuration Manager –SSRS
Generating the Report
SQL Server 2005 – SQL Server Business Intelligence Development Studio
Step 1:
Click on File –New Project – Report Server Project Wizard
Name: User Report
Location: C:\TestReport
Click Ok & Click Next
Step 2:
New Data Source
Name: Datasource1
Type: MS SQL Server
Connection String
Click on Edit
Server Name: Sys-PC (Server Name of the SQL Server)
Database Name: TempDB (Select the User Defined Database)
Click Test Connection and Click Ok
Click Next
Step 3:
Query String

```
SELECT ENAME, JOB, SAL FROM EMP
```

Click Next
Step 4:
Select Report Type—Tabular
Click Next
Step 5:

Design the Table

Groups – Job

Details – Ename, Sal

Click Next

Step 6:

Choose the Table Style—Slate

Step 7:

Choose the Deployment Location

Report Server: http://localhost/ReportServer

Deployment Folder: Report Test

Click Next

Step 8:

Completing the Wizard

Check the Preview Report

Click Finish