



INDEX

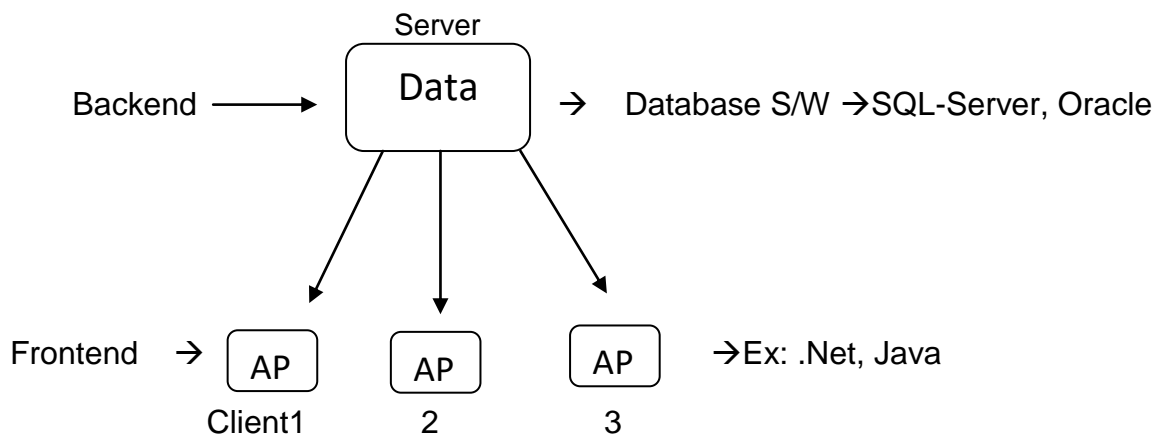
1) SQL Constraints	2
2) Select Statement	22
3) Joins & Join Tables	25
4) Sub Queries	30
5) Built in Functions	33
6) Operators	37
7) Stored Procedures	40
8) Triggers	42
9) UDF's	43
10) Cursor	44
11) Views	46
12) SQL Server Security	49
13) Normalization & PPT	55



SQL Constraints

Client-Server Architecture:

- It is the most frequently used architecture model in developing the projects. In this model there exists a main computer called server which is connected with multiple other computers called clients.
- The server will be used to store the data in an organized manner with the help of data based Software. Such as SQL-Server, oracle etc... This data will be used by the application program available on client machine.



Database Technologies:

The database principles are practically implemented by using the following technologies.

DBMS → Database Management System

RDBMS → Relational Database Management System

ORDBMS → Object Relational DBMS

Differences between DBMS & RDBMS

DBMS	RDBMS
1) It is a Single User System.	1) It is a Multiuser System.
2) We can't provide Security to the Table.	2) We can establish full security to the Data & Operations based on the Data. Eg: SQL-Server, Oracle, My SQL

Note: RDBMS is having a problem of not containing reusability. It is avoided in ORDBMS with the introduction of object oriented concepts.

Ex: Latest versions of SQL-Server & Oracle.



SQL-Server:

This database s/w is developed by Microsoft organization. Like any other Microsoft component it can also be used only on windows platform. It comes with less cost and more facilities when compared with competitive database oracle.

Versions of SQL-Server:

- Ms SQL-Server 6.0
- Ms SQL-Server 6.5
- Ms SQL-Server 7.0
- Ms SQL-Server 2000 (8.0)
- Ms SQL-Server 2005 (9.0)
- Ms SQL-Server 2008 (10.0)
- Ms SQL-Server 2008 R2 (10.5)
- Ms SQL-Server 2012 (11.0)
- Ms SQL-Server 2014 (12.0)

SQL-Server Management studio tool:

It is a GUI tool used to perform database operations in SQL-Server.

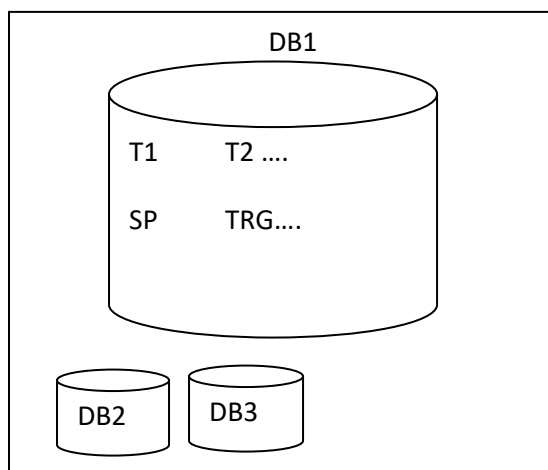
→ It contains a window called "Object explorer" where we can perform operations only by following navigations.

→ By clicking on the new query icon of the tool bar we will get a query editor window where we can execute related SQL commands to perform the operation.

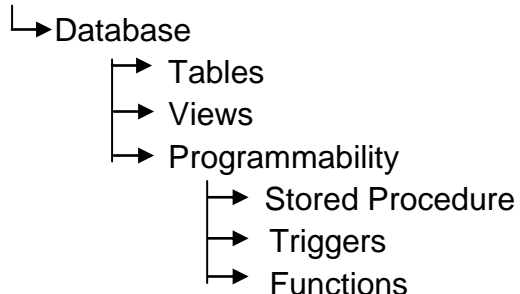
Components/Architecture of SQL-Server:

Whatever version of SQL-Server we are using all will follow the same architecture model as specified below.

Server



Server





While working with SQL-Server we have to perform the following operations.

- 1). Establish a connection with local or remote instance of SQL-Server.
- 2). Create a location called db in the connected server.
- 3). Creating an object called table inside the db to store actual data.
- 4). Create and use all other objects such as
 - Stored procedures
 - Triggers

Connecting to server:

We can establish a connection with SQL-Server in 2 ways.

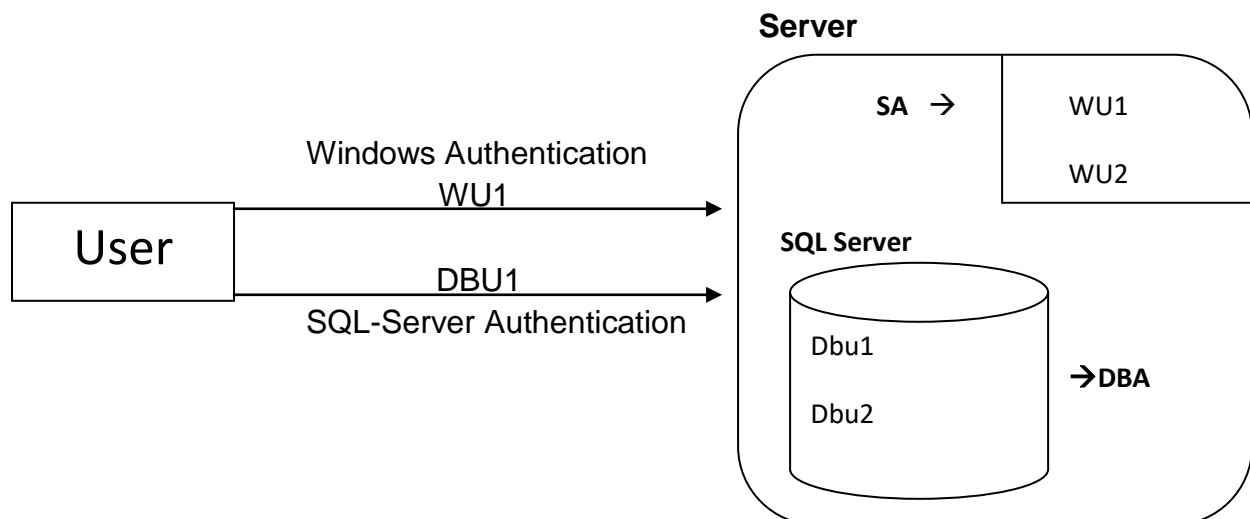
- Windows Authentication
- SQL-Server Authentication

1) Windows Authentication:

In this method we can use a windows login a/c to connect to the server. These users will be created by sys admin.

2) SQL-Server Authentication:

In this case we can use a DB User Account to establish the connection. These Account's will be created by DBA.





Data Bases:

A data base is a location where we can store the data in an organized manner.

SQL-Server provides 2 types of databases. They are:

- System DB
- User defined DB

1) System DB:

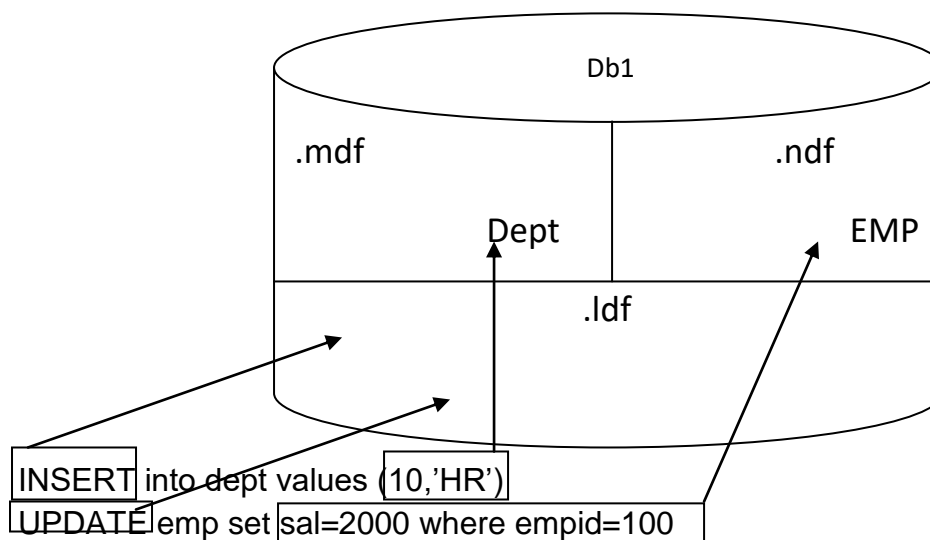
Along with the installation of SQL-Server by default 4 databases will be created automatically called as system db. They will be used internally by SQL-Server for its correct functionality.

They are:

- Master
- Model
- MSDB
- Temp DB

2) User defined db:

- These are the db's that are created by the user. In order to access these db's we must have related permission from the admin or db owner.
- In the SQL-Server a db is a logical combination of some physical files called data files. In some of the files result of the command will be stored in some other files details about the command will be stored. They are:
 - Primary data file (.mdf- Master Data File)
 - Secondary data file (.ndf- New Data File)
 - Transaction log file (.ldf-Log Data File)
- Primary or secondary file can be used to store result of the command and log file will be used to store details about the command.
- Every database must contain at least 1 primary file and one log file. Creating secondary file is optional.





Creating a data base using navigation:

- 1).Open the management studio tool and connect to the required server.
- 2).In object explorer window right click on data bases folder and select "New data base" option.
- 3).Provide a name to the data base.
- 4).Specify the primary and log file details.
- 5).Click OK button.

Creating data base using command:

Syn1: CRATE DATABASE <data base name>

Ex: create database db1

Syn2:

```
1) CREATE DATABASE <database Name>
2) [ON
3) (Name='<logical filename>',
4)  Filename='<OS filename>',
5)  SIZE=<in MB's>)
6) LOG ON
7) (Name.....)]
```

Note:

- 1). "OS filename" specifies location of th file in the server.
- 2). "Logical filename" is the duplicate name for the location.
- 3). '< >' represents a name/value.
- 4). [] represents optional parameters.

```
1) create database dbInsure
2) on
3) (name='dbinsuredata',
4)  filename='E:\mydbfiles\dbinsure_data.mdf',
5)  size=20)
6) log on
7) (name='dbinsurelog',
8)  filename='E:\mydbfiles\dbinsure_log.ldf',
9)  size=10)
```

Adding secondary file to database:

```
1) alter database dbInsure
2) add file
3) (name='dbinsuredata1',
4)  filename='E:\mydbfiles\dbinsure_data1.ndf',
5)  size=20)
```



SP_HELPDB:

This system stored procedure can be used to get the description of specified data base name. If we are not providing any name of the data base. It returns the list of all data bases in the connected server.

Tables:

A table is a database object used to store the data in the form of rows and columns. It is the only format user can store and get back the data.

Based on the type of data we are storing SQL-Server provides 3 types of tables. They are:

- System Tables
- User tables
- Temporary Tables

1) System Tables:

These tables will be created automatically along with the database creation. They will contain system level information.

Ex: Sys objects, Sys users etc

Note:

To get the list of all user tables in current database use the following command.

→ select name from sys objects where xtype='U'

2) User Tables:

These are the tables that are created by the user. In order to access these tables we must have related permission from the owner of the table or DBA. They will be created permanently in the hard disk of server.

3) Temporary Tables:

These tables are also created by the user, but they will be created temporarily in the buffer. They will be identified by # symbol.

Creating a Table:

While creating a table either by following navigations or commands we must specify the following details.

- Name of the table
- Details of columns such as column name, data type, and size.



Data Types:

A data type will specify the type of data to be stored in a column. The most frequently used data types in SQL-Server are:

1).int: It is used to store numeric data of whole numbers. By default it occupies 4 bytes of memory.

Ex: empid int

2).decimal: It is used to store numeric data of floating point numbers.

Ex: unit price decimal (5, 2)
123.45

3).char: It is used to store alpha numeric and special characters of fixed length. In this case remaining unused memory will be wasted.

Ex: name char (10)

M	A	H	E						
---	---	---	---	--	--	--	--	--	--

4).varchar: It is also used to store alpha numeric & special characters but with variable length. In this case the unused memory will be released to SQL-Server for further usage.

Ex: name varchar (10)

M	A	H	E						
---	---	---	---	--	--	--	--	--	--

5).date: It is used to store a date either in 'mm/dd/yy', or 'mm-dd-yy' format. By default it occupies 8 bytes of memory.

Ex: DOJ date

6).date time: It used to store date along with time. It also occupies 8 bytes of memory.

→The following are other data types supported by SQL-Server.

Bit, tiny int, small int, big int, text, nchar, nvarchar, ntext, image, sql variant, table, XML.

Note:

The following are the maximum capacity specifications for a table.

- 1).Table name or column name can contain max of 255 characters including spaces.
- 2).A table can contain a max of 1024 columns.
- 3).The max size of data that can be entered in a single record of a table is 8060 bytes.

Creating a Table Using Navigation:

- 1).Open the management studio tool and connect to the required server.
- 2).Navigate up to the db where we want to create the table.
- 3).Inside that db right click on tables and select new table option.
- 4).In the window displayed specifies the column details such as column name, data type, & size.
- 5).Click on the save table icon to provide a name to the table.



Modifying Table Structure:

- In the object explorer window right click on table name and select design option.
- In the window displayed add new columns, modify or delete existing column data.
- Click on save table icon to save the changes.

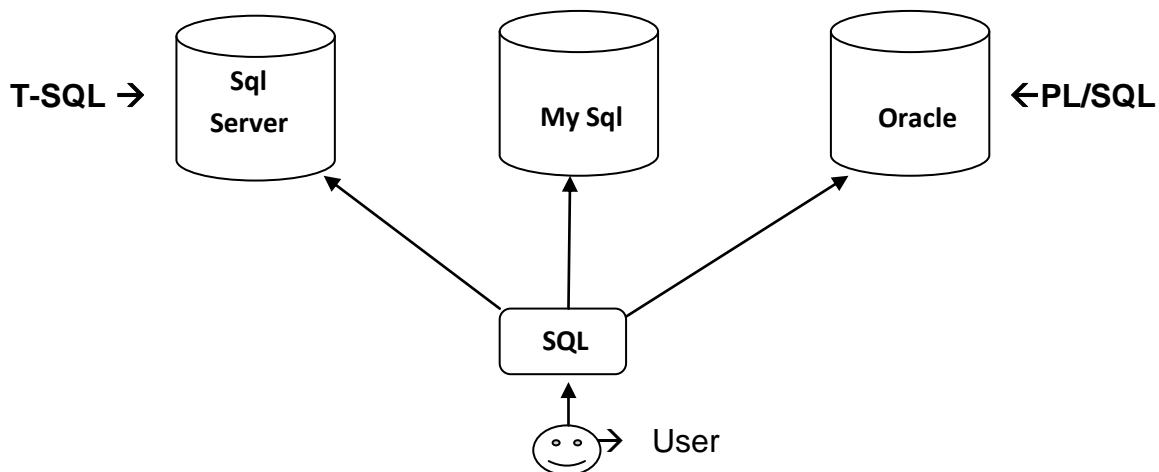
Manipulating Table Data:

- In the object explorer window right click on the table name and select Edit top 200 rows option.
- 2). In the window displayed add new records, modify or delete existing record data.

Note: No need to save the changes when we close window changes will be saved automatically.

SQL:

- ⊕ It stands for structured query language. It is a language that contains a set of standards in the form of commands that should be followed by all database s/w. It means all SQL commands should be executable in all the database s/w.
- ⊕ Along with these sql commands every db contains their own programs that are executable in the db only.



Types of SQL-Commands:

Based on the type of operations we are performing SQL commands are divided into the following ways.

- DDL (Data Definition Language)
- DQL (Data Query Language)
- DML (Data Manipulation Language)
- TCL (Transaction Control Language)
- DCL (Data Control Language)



1) DDL:

This language commands will be used to create a new db object, modify or delete an existing object from the server. All these commands will effect the structure of the object directly.

CREATE DROP
ALTER TRUNCATE

Note:

Except TRUNCATE the remaining DDL commands are called as auto commit commands. It means with the execution of these commands result will be saved directly into db. We can't undo the changes.

2) DQL:

This language command will be used to retrieve the data from the table.

Ex: SELECT

3) DML:

This language commands will be used to perform operations on the table data. With these commands we can able to enter, modify or delete the data.

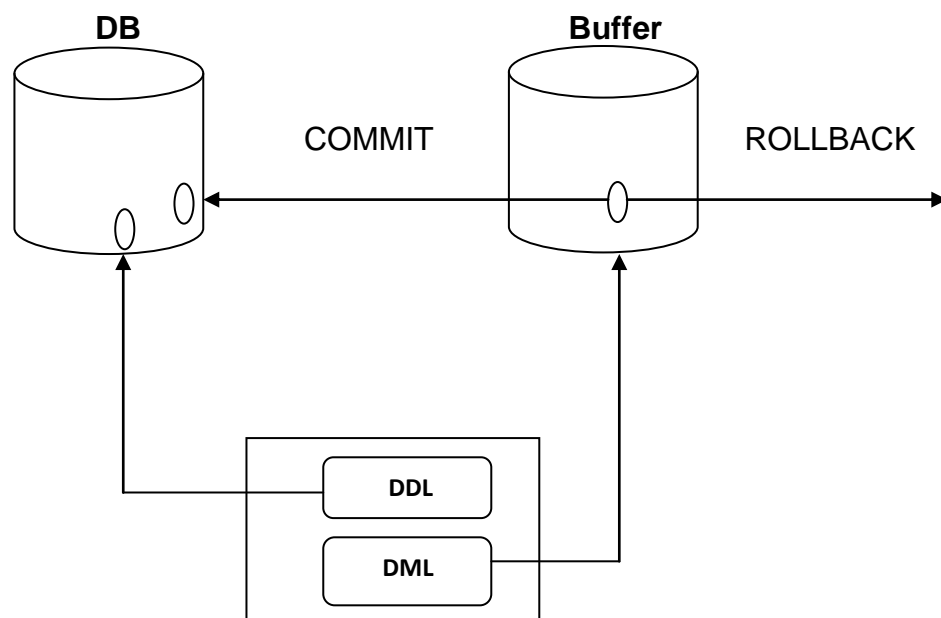
INSERT UPDATE DELETE

4) TCL:

When we perform a DDL operation result will be stored directly into db. Where as if we execute a DML operation such as INSERT, DELETE, UPDATE at first changes will be saved into the buffer. We can control these changes by using TCL commands.

COMMIT ROLLBACK

COMMIT will be used to save the changes and ROLLBACK will be used to remove the changes.





5) DCL:

This language commands used to control the security.

GRANT REVOKE

GRANT is used to provide permission.

REVOKE is used to remove the permission.

Creating a Table Using Command:

Syn:

- 1) **CREATE TABLE** <table name>
- 2) (<col1><data type><size>[**NOT NULL** | **DEFAULT** <constant value>
- 3) | **IDENTITY**(seed,increment)],<col2>....)

NOT NULL:

A column with this option doesn't allow NULL values. The user must provide a value to the column while entering the table.

DEFAULT:

The constant value specified in this column will be used as the column data, when the user is not providing any value at the time of insertion.

Ex:

- 1) **create table** products
- 2) (proid **int** **NOT NULL**,
- 3) proname **char**(10),
- 4) proqty **int** **DEFAULT** 500)

IDENTITY:

This property will be used to generate a number automatically in the column. The auto generation can be controlled with seed and increment values.

Seed specifies initial value of the column.

Increment specifies how much the previous value should be incremented for each insert.

A table must contain only one identity property. It must be specified on a numeric column only.

Ex:

- 1) **create table** customers
- 2) (custid **int** **IDENTITY**(10,1),
- 3) custname **char**(10))



SP_HELP:

This system stored procedure can be used to get the description of specified db object. (Ex: Table) If we are not providing any object name this command returns the list of all objects in current db.

Syn: SP_HELP<object name>

Ex: sp_help products
Sp_help

Modifying Table Structure:

The alter command can be used to perform modifications to the table structure.

Syn:

- 1) ALTER TABLE<table name>
- 2) ADD<column name><data type>...
- 3) | ALTER COLUMN<column specification>
- 4) | DROP COLUMN <column name>

Ex: Adding a column:

- 1) alter table customers
- 2) add custadd varchar(30)

Modifying a column:

- 1) alter table customers
- 2) alter column custname varchar(20)

Deleting a column:

- 1) alter table customers
- 2) drop column custadd

Deleting a Table:

The DROP command can be used to remove the table from server.

Syn: DROP TABLE <TABLE NAME>

Ex: drop table emp



Entering Table Data:

The INSERT command will be used to enter the data into the table. When we are entering all the column values in the same order then it is optional to specify column list, where as if we are entering particular column values only then we must specify the column list by representing the names of columns for which we are specifying the data.

Syn: INSERT INTO <table name>(<column list>)VALUES(<COLUMN DATA>)

Ex:

- 1) insert into products(proid,praname,proqty)values(1,'p1',100)
- 2)
- 3) insert into products values(2,'p2',200) ◇No need to specify column list.
- 4)
- 5) insert into products values(3,'p3') ◇Error must specify the column list.
- 6)
- 7) insert into products (proid,praname) values (3,'p3')
- 8)
- 9) insert into products values (4,'p4',200),(5,NULL,600),(6,'p6',100)
→ Entering multiple records at a time.

Note: Entering multiple records in a table is called as bulk insertion.

Modifying Table Data:

The UPDATE command is used to modify the data of an existing table. We can able to modify a single row or column and multiple rows or columns in the table.

Syn: UPDATE <table name> SET <col1>=<val1>,<col2>=<val2>...
WHERE <condition>

Ex:

- 1) update products set proqty=50
- 2) → Modify proqty value for all records.
- 3)
- 4) update products set proqty=100 where proid=3
- 5) → modify product value only for product 3.
- 6)
- 7) update products set praname='p5',proqty=150 where proid=5
- 8) → modify praname,proqty values for proid 5.
- 9)
- 10) update products set proqty=200 where proid=1 or proid=4
- 11) → modify provalue for records with proid 1/4.
- 12)
- 13) update products set proqty=300 where proid IN(1,2,5,6)
- 14) → modify provalue for records with proid 1/2/5/6.



Deleting Table Data:

Either by using DELETE or TRUNCATE we can able to remove the data from the table. Both these commands will remove only the table data without effecting the structure.

TRUNCATE:

Syn: TRUNCATE TABLE <table name>

Ex: truncate table products

DELETE:

Syn: DELETE FROM <table name> WHERE <condition>

Ex: 1).delete from products.

2).delete from products where proid=2

Difference between Delete & Truncate:

Delete	Truncate
1) It can remove either all records or specific record.	1) It can remove all the records of the table only
2) Even though we are deleting all the records the deletion will be done row by row.	2) In this case all the records will be removed at a time. So it executes faster than the delete command.
3) After deleting all the records when we insert a new record in the IDENTITY column auto generation will be continued normally with the next higher number.	3) In this case auto generation will be restarted from the beginning.
4) It maintains removed record details in log file.	4) It maintains removed data page details in log file.

Data Page:

It is a format of storing the data inside SQL-Server. As a user we will store and get back the data in the form of tables. All these table data inside SQL-Server will be stored in the form of "data pages". Each data page will occupy 8kb of memory. A combination of 8 pages is called as 1 "Extent". If the table data is not fit in a single page then multiple pages will be occupied by the table.

Note:

- In case of oracle db after deleting the records with TRUNCATE command we can't get back the removed records. But after DELETE command we can roll back the removed records by using ROLLBACK.
- Whereas in case of SQL-Server db either after DELETE or TRUNCATE we can roll back the deleted records.



CONSTRAINTS:

- A constraint or an integrity constraint is a restriction that can be applied on a column of a table to satisfy customer requirements.
- The process of maintaining correctness of data among the tables according to customer requirements is called as data integrity.
- As a SQL developer it is our responsibility to apply related constraints of the columns of table.

Types of Constraints:

Based on the type of operations we are performing constraints are divided into following ways.

1) Domain Integrity Constraint:

It is used to provide a restriction to the column data.

Ex: CHECK

2) Entity Integrity Constraint:

These constraints will be used to avoid the entry of duplicate values in a column.

Ex: UNIQUE, PRIMARY KEY

3) Referential Integrity Constraint:

It is used to establish relationship b/w the tables.

Ex: FOREIGN KEY

CHECK Constraint:

A column with this constraint will contain a condition only those values that satisfies the condition will be allowed into the column.

Syn:

- 1) **CREATE TABLE** <table name>
- 2) (<col1><data type>(<size>)[**CONSTRAINT**<constraint name>]
- 3) **CHECK**(<condition>),<cl2>...)

Ex:

- 1) **create table** emp (eid **int**, ename **char**(10), dno **int**, sal **int** **CONSTRAINT** CK_EMP_SAL **CHECK** (sal between 10000 and 50000)
- 2)
- 3) **insert into** emp **values**(100,'E1',20,25000)
- 4)
- 5) **insert into** emp **values**(101,'E2',10,5000)
- 6) → Error sal value not satisfying the condition.



Deleting a check constraint:

Syn:

```
ALTER TABLE<table name>  
DROP CONSTRAINT CK_EMP_SAL
```

Ex:

```
alter table emp  
drop constraint CK_EMP_SAL
```

Adding a check constraint:

If the table already contains the data then the condition to be apply must satisfied existing column data.

Syn:

```
ALTER TABLE<table name>  
ADD CONSTRAINT <constraint name>  
CHECK <condition>
```

Ex:

- 1) `insert into emp values(101,'E2',10,5000)`
- 2) → Correct since remove `constraint` in previous.
- 3)
- 4) `alter table emp`
- 5) `add constraint CK_EMP_SAL`
- 6) `check(sal between 10000 and 50000)`
- 7) → Error since `some of values` are not satisfying condition.
- 8)
- 9) `update emp set sal=30000 where sal<10000 or sal>50000`
- 10)
- 11) `Execute` command `no 2`. Now it will `execute` successfully.

RULES:

Like a check constraint it can also be used to apply a condition to a column.

The difference between CHECK & RULES are:

CHECK	RULES
1) It can be applied on a specific column of a particular table.	1) It is a separate db object that can be binded to any column of any table that satisfies the data type.
2) While adding the CHECK constraint it considers existing data. When it satisfies the condition then only constraint can be applied.	2) It ignores existing data and considers only new data.



Syn:

CREATE RULE <rule name>
AS <condition>

Ex:

- 1) create table stock (ino int, idesc char(10), minqty int, maxqty int)
- 2)
- 3) insert into stock values (100, 'I1', 100, 500), (101, 'I2', 150, 1000), (102, 'I3', 50, 600)
- 4)
- 5) select * from stock
- 6)
- 7) create rule myrule as @val >= 100

Binding a Rule:

Syn: SP_BINDRULE '<rule name>', '<table.column>'

- 1) sp_bindrule 'myrule', 'stock.minqty'
- 2)
- 3) insert into stock values (103, 'I4', 200, 800)
- 4)
- 5) insert into stock values (104, 'I5', 20, 800) → Error

Unbinding Rule:

Syn: SP_UNBINDRULE '<table.column>'

Ex: sp_unbindrule 'stock.minqty'

Deleting a Rule:

Syn: DROP RULE <rule name>

Ex: drop rule myrule



UNIQUE Constraint:

- A column with this constraint doesn't allow duplicate values but allows a NULL value.
- A table can contain more than one UNIQUE constraint.
- In case of SQL-Server unique allows only one NULL value where as incase of oracle UNIQUE allows more than one NULL value.

Syn:

```
CREATE TABLE <TableName>
(<ColName1> <datatype> [(size)]
[CONSTRAINT <ConstraintName>] UNIQUE,
<ColName2>.....)
```

Ex:

- 1) `create table dept (dno int constraint uk_dept_dno unique, dname char(10))`
- 2)
- 3) `insert into dept values(10,'HR')`
- 4)
- 5) `insert into dept values(10,'ADMIN')` → Error, duplicate dno value
- 6)
- 7) `insert into dept(dname) values('Tester')`
- 8)
- 9) `insert into dept(dname) values('Dev')` → Error, duplicate NULL value into dno column

Adding a UNIQUE constraint:

Syn:

```
ALTER TABLE <table name>
ADD CONSTRAINT <constraint name>
UNIQUE (<column name>)
```

Ex:

```
alter table dept
add constraint UK_DEPT_DNO
unique(dno)
```

PRIMARY KEY:

- A column with this constraint doesn't allow duplicate values as well as NULL values.
- A table must contain only one PK. It must be specified on a column based on which we can able to identify remaining column data.



Ex:

- 1) `create table products`
(`prodid int constraint PK_Products_Prodid primary key`, `prodname char(10)`, `proqty int`)
- 2)
- 3) `insert into products values(1,'P1',100)`
- 4)
- 5) `insert into products values(1,'P2',200)` → Duplicate `prodid`.
- 6)
- 7) `insert into products values(null,'P3',250)` → Error can't enter `NULL values`.

Composite PK:

If it is not possible to specify a PK on any single column of the table then check the table data to identify any column combination that can be used to find the other column data. If it is available make that combination as a PK called as composite PK. A maximum of 16 columns can be combined in a composite key.

Order Details:

[ONO	INO]	ODATE	CNO	QTY
-----	-----	-----	-----	-----
100	I1	1/1/14	20	25
100	I2	1/1/14	20	30
101	I3	1/2/14	10	25
101	I2	1/2/14	10	20
102	I1	1/2/14	20	25

Note:

- If we provide a constraint beside a column is called column level constraint. Whereas if we specify constraint at the end of all columns then it is called as table level constraint.
- If the constraint is based on one column then we can specify the constraint either at column level or at table level. Whereas if the constraint is based on more than one column then we must specify table level constraint.

Differences between UNIQUE & PRIMARY KEY

UNIQUE	PRIMARY KEY
1) It allows a null value.	1) It doesn't allow null values.
2) A table can contain more than one unique constraint.	2) A table must contain only one primary key.
3) By default a non-clustered index will be created on unique column.	3) By default a clustered index will be created on the column.



FOREIGN KEY:

- This constraint is used to establish relation between the tables. The table with FK is called as Referencing/Child table. The table with PK is called as Referenced/Parent table.
- Once we establish a relation between tables some restrictions will be applied to the db operations. They are:
 - 1) The data to be entered in FK column of the child table must have a matching value in the PK column of the parent table.
 - 2) It is not possible to directly delete a record from the parent table, when it is having references in the child table. In this case at first we have to remove all the child table references along with parent table records.

Syn:

```
1) CREATE TABLE <TableName>
2) (<ColName1> <Datatype> [(Size)]
3) [CONSTRAINT <ConstraintName>] FOREIGN KEY
4) REFERENCES <ParentTable>(<PrimarykeyCol>)
5) [ON DELETE CASCADE],
6) <ColName2>.....)
```

Ex:

```
1) create table orders (ordno int,orderdate datetime,custno int,
   prodid int constraint fk_orders_prodid FOREIGN KEY REFERENCES products(prodid),qty int)
2) select * from products
3)
4) insert into products values (2,'P2',200),(3,'p3',250)
5)
6) select * from products
7)
8) insert into orders values (100,'1/1/14',20,3,50)
9)
10) insert into orders values (101,'1/2/14',10,5,25)
    →error, prodid is not having reference in parent table.
11)
12) insert into orders values
    (101,'1/2/14',10,2,25), (102,'1/3/14',20,1,20),(103,1/4/14',30,2,60)
13)
14) delete from products where prodid = 2
15) → Error, since this record is having some reference in child table.
```



ON DELETE CASCADE:

In general it is not possible to directly delete a record from the parent table when it has any reference in the child table. Whereas with the use of ON DELETE CASCADE in the syntax of FK whenever we delete the parent table record that record and all its child table references will be removed at a time.

Syn:

```
ALTER TABLE <TableName>
ADD CONSTRAINT <ConstraintName>
FOREIGN KEY(<ColName>)
REFERENCES <ParentTable>(<PKColumn>)
ON DELETE CASCADE
```

Ex:

- 1) `alter table orders`
`drop constraint fk_orders_prodid`
→ removing existing `constraint`
- 2)
- 3) `alter table orders`
`add constraint fk_orders_prodid`
`foreign key(prodid)`
`references products(prodid)`
`on delete cascade`
→ adding the `foreign key with new option`
- 4)
- 5) `delete from products where prodid=2`
→ Deletes the records `with prodid 2` both `from products and orders tables`.

ON UPDATE CASCADE:

Whenever we modify the data in the PK column of the parent table if it has to reflect in the FK column then we have to make use of ON UPDATE CASCADE in the syntax of FK.

Note: A table can contain maximum of 253 references.

Note: Even though the column doesn't contain NULL's, duplicates we can't add PK. At first we have to provide a NOT NULL option to the column & then add the PK



SELECT Statement

Creating a table:

```
CREATE TABLE emp  
(empid int, Ename char(10), Sal int, dno int)
```

Insert Data:

```
INSERT INTO emp VALUES (1,'E1',25000,20), (2,'E2',30000,10),  
                        (3,'E3',20000,20), (4,'E4',15000,10), (5,'E5',30000,30)
```

Retrieving Table Data:

The SELECT statement is used to retrieve the data from the table. It is possible to retrieve all the table data or particular column data. In order to retrieve complex data from the tables, we can add extra optional clauses to the select statement.

Note: A maximum of 4096 columns can be specified in a select statement.

Syn:

```
SELECT <column list> FROM <tablename>  
[WHERE <condition>  
[GROUP BY <column name>  
[HAVING <Condition based on the grouped data>  
[ORDER BY <column name> [ASC | DESC ]
```

- 1) Select * from emp
- 2) Select empid,ename,sal,dno from emp
- 3) Select count(*) from emp

WHERE Clause:

It is used to provide a condition in the select statement. Only those records that satisfy the condition will be displayed as output. The condition can be specified based on any column of the table.

Q: Get the details of employees who belong to the department 20 and also having more than 20000 salary.

Sol: Select * from emp
 Where dno = 20 AND sal > 20000



Output:

Empid	ename	sal	dno
1	E1	25000	20

GROUP BY Clause:

It is used to group the table data based on a column. After grouping the table data, we can able to apply an aggregate function on each group independently. The aggregations include: COUNT, MIN, MAX, AVG and SUM.

Ex:

Q.1: Get the no.of employees in each department.

Sol: SELECT dno, count(*) from emp GROUP BY dno

Output:

dno	NoOfEmployees
10	2
20	2
30	1

Q.2: Get the total amount paying to each department

Sol: SELECT dno, SUM(sal) from emp GROUP BY dno

Output:

dno	TotalSalary
10	45000
20	45000
30	30000

HAVING Clause:

- Like a 'Where' clause, HAVING clause is also used to provide condition based on the grouped data. The difference between them is :
- With WHERE clause, condition can be specified on entire column data at a time. i.e., before grouping the table data. Whereas, with HAVING clause the condition can be specified after grouping the table data on each group independently.
- When to use which clause depends on the criteria of the query. If the criteria doesn't require grouping, use where clause other wise use having clause.

Examples:

Q.1: Get the no.of employees in each department with employees having more than 25000

Sol: SELECT dno, Count(*) NoOfEmployees from emp WHERE sal > 25000 GROUP BY dno



Output:

dno	NoOfEmployees
10	1
30	1

Q.2: Get the department which are having more than one employee

Sol: SELECT dno, Count(*) NoOfEmployees from emp GROUP BY dno HAVING count(dno) > 1

Output:

dno	NoOfEmployees
10	2
20	2

ORDER BY Clause:

It is used to make the table data into different orders based on a column. The default storing order is ascending to get the output in descending order use DESC key word.

Ex: Get the employee details in alphabetical order.

Select * from emp order by ename.

Note: It is also possible to sort the table data based on more than one column. In this case at first all the table data will be sorted based on first column normally, then for each value group of first column related second column data will be sorted.

Ex: select * from emp orderby sal desc.



JOINS

- A Join is a SELECT statement used to retrieve the data distributed among different tables.
- In order to join the tables, they must have a common column. Common column means the names of the columns can differ, but they must have the same data type, size and data.

Syntax of a Join:

```
SELECT <ColumnList>  
FROM <LeftTable> <JoinType> <RightTable>  
ON <criteria>
```

Types of Joins:

Based on the way the select statements are provided and the records they return, joins are categorized into the following ways.

- 1) Inner Join
- 2) Outer Join
- 3) Left Outer Join
- 4) Right Outer Join
- 5) Full Outer Join
- 6) Cross Join
- 7) Self Join

1) Inner Join:

This join returns only the matching data between the tables. It is mainly used to retrieve the related data between the tables.

Ex: *Get the details of the students along with the class names from the following student and class tables.*

Class:

Student:

CID	CNAME	CSIZE	SID	SNAME	CID	MARKS
100	MCA	30	1	A	101	300
101	MTECH	25	2	B	100	200
102	BTECH	40	3	C	101	150
103	BCA	30	4	D	102	150
			5	E	NULL	300

Sol:

- 1) **SELECT** sid,sname,s.cid stdCID,c.cid clsCID,cname
- 2) **FROM** Student s **INNER JOIN** Class c
- 3) **ON** s.cid=c.cid



Output:

sid	sname	stdCID	clsCID	cname
1	A	101	101	MTECH
2	B	100	100	MCA
3	C	101	101	MTECH
4	D	102	102	BTECH

2) Outer Joins:

This join returns the unmatched records between the tables. Which tables unmatched data to be displayed, depends on the type of outer join we are using. Sometimes the unmatched data will be displayed along with matched data and some other times the unmatched data will be displayed alone.

3) Left Outer Join:

This join returns the unmatched data from the left table. The corresponding values from the right table will be NULL.

Ex:

a. Get the details of all students along with classes if any student is having a class.

Sol:

```
SELECT sid,sname,s.cid stdCID,c.cid clsCID,cname  
FROM Student s LEFT OUTER JOIN Class c ON s.cid=c.cid
```

Output:

sid	sname	stdCID	clsCID	cname
1	A	101	101	MTECH
2	B	100	100	MCA
3	C	101	101	MTECH
4	D	102	102	BTECH
5	E	NULL	NULL	NULL

b. Get the details of students who are not assigned to any class.

Sol:

```
SELECT sid,sname  
FROM Student s LEFT OUTER JOIN Class c ON s.cid=c.cid WHERE c.cid is null
```

Output:

sid	sname
5	E



4) Right Outer Join:

This join returns the unmatched data from the right table. The corresponding values from the left table will be null.

Ex:

a. *Get the details of all classes along with students.*

Sol:

```
SELECT sid,sname,s.cid stdCID,c.cid clsCID,cname  
FROM Student s RIGHT OUTER JOIN Class c ON s.cid=c.cid
```

Output:

sid	sname	stdCID	clsCID	cname
2	B	100	100	MCA
1	A	101	101	MTECH
3	C	101	101	MTECH
4	D	102	102	BTECH
NULL	NULL	NULL	103	BCA

b. *Get the details of classes which are not having students.*

Sol:

```
SELECT c.cid,cname  
FROM Student s RIGHT OUTER JOIN Class c ON s.cid=c.cid WHERE s.cid is null
```

Output:

cid	cname
103	BCA

5) Cross Join:

A join without common column criteria is called as a cross join. This join is used to find the different combinations of data between the tables. The output will be the product of number of records between the tables.

Ex: Get the different possibilities how a student can be assigned to a class only when they are having more than 200 marks

Sol:

```
Select sid, sname, c.cid, cname  
From student s cross join class c where marks>200
```

NOTE:

In case of ORACLE database, Inner join is called as EQUI/NATURAL Join and Outer Join is called as Non-EQUI Join.



6) Self Join:

- Joining a table to it self is called as a self-join. In a self-join, since we are using the same table twice, to distinguish the two copies of the table, we will create table aliases.
- In the syntax of self join we never use the keyword 'self'. Internally, we perform an inner join only.

Ex: Get the employee details along with manager names from the EMP table. In this table one the employee will be the manager for other employee.

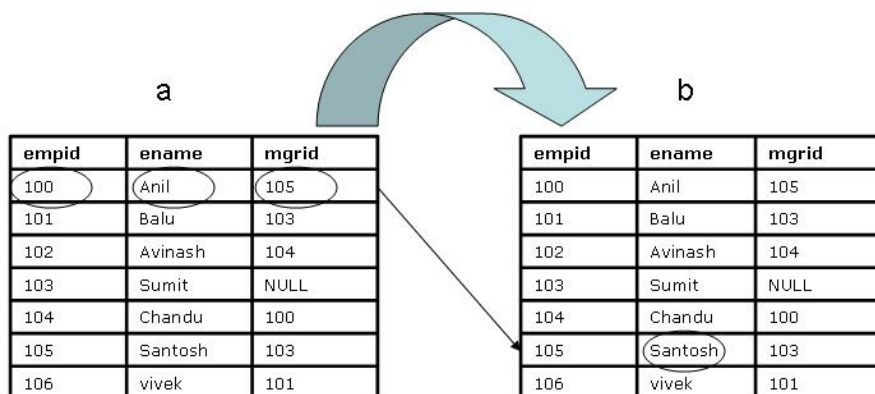
Sol:

```
Select a.empid,a.ename,a.mgrid,b.ename mgrName  
From emp a inner join emp b On a.mgrid=b.empid
```

Output:

empid	ename	mgrid	MgrName
101	B	104	E
102	C	103	D
103	D	101	B
104	E	100	A

Internal Execution Process:





Case Study:

DEPT Table:

Dno	Dname
10	HR
20	ADMIN
30	TESTER
40	DEV

EMP Table:

Empid	Ename	Mgrid	Dno	Sal	Com	Hiredate
100	Anil	105	20	15000	10	1/12/93
101	Balu	103	10	35000	NULL	4/6/92
102	Avinash	104	30	39000	20	2/5/93
103	Sumit	NULL	10	38000	10	10/11/90
104	Chandu	100	20	50000	30	1/1/93
105	Santosh	103	10	30000	NULL	4/5/94
106	Vivek	101	20	35000	15	6/7/00

Ex:

- 1). Get the emp details along with dnames from dept & emp.
- 2). Get the details of all employees along with & without employees.
- 3). Get the details of depts. Which are not having employees?
- 4). Get the possibilities about assign dept to an employee.
- 5). Get the employee details along with mgr names from emp.



SUBQUERIES

A SubQuery is a SELECT statement containing another SELECT statement. In these SubQueries, at first, the inner query will be executed and based on the result next higher query will be executed. Whenever, it happened to compare a column data with the output of another select statement then we use sub queries.

Types of SubQueries:

Based on the way the Select statements are provided and the way they will execute, SQL Server provides the following types of SubQueries.

- 1) Simple SubQuery
- 2) Nested SubQuery
- 3) Multiple SubQuery
- 4) Correlated SubQuery

1) Simple SubQuery:

It is a select statement containing another select statement. At first inner query will be executed and based on the result next higher query will be executed.

Ex: *Get the Second maximum salary from the following EMP table.*

Empid	Ename	Mgrid	Sal	Comm	Hiredate
100	Anil	105	15000	10	1/12/93
101	Balu	103	35000	NULL	4/6/92
102	Avinash	104	39000	20	2/5/93
103	Sumit	NULL	38000	10	10/11/90
104	Chandu	100	50000	30	1/1/93
105	Santosh	103	30000	NULL	4/5/94
106	Vivek	101	35000	15	6/7/00

Sol: `SELECT max(Sal) FROM EMP where sal<(SELECT max(Sal) FROM EMP)`

Output: 39000

2) Nested SubQueries:

A Nested SubQuery is a select statement, containing another select statement, which contains another select statement and so on. Such type of nesting nature of select statements is called as a nested subquery. In these SubQueries, at first, the innermost query will be executed and



based on the result next higher query will be executed and so on. A maximum of 32 select statements can be combined in these nested SubQueries.

Ex: *Get the Third maximum salary from the emp table*

Sol:

```
SELECT max(sal) FROM EMP WHERE sal<(SELECT max(sal) FROM EMP  
WHERE sal<(SELECT max(sal) FROM EMP))
```

Output: 38000

3) Multiple SubQueries:

It is select statement containing different select statements at different places.

Ex: *List out the employees who are having salary less than the maximum salary and also having hire date greater than the hire date of an employee who is having the maximum salary.*

Sol:

```
SELECT empid,ename,sal,hiredate FROM EMP  
WHERE sal<(SELECT max(sal) FROM EMP) AND  
hiredate>(SELECT hiredate FROM EMP WHERE sal=(SELECT max(Sal) FROM EMP))
```

Output:

Empid	Ename	Sal	Hiredate
100	Anil	15000	1993-01-12
102	Avinash	39000	1993-02-05
105	Santosh	30000	1994-04-05
106	vivek	35000	2000-06-07

4) Correlated SubQueries:

In a normal SubQuery, at first, the inner query will be executed and only once. Based on the result next higher query will be executed.

Where as in a correlated SubQuery, the inner query will be executed for each record of the parent statement table. The internal execution process of this SubQuery will be as follows:

- 1) A record value from the parent table will be passed to the inner query.
- 2) The inner query execution will be done based on that value.
- 3) The result of the inner query will be sent back to the parent statement.
- 4) The parent statement finishes the processing for that record.

The above 4 steps will be executed for each record of the parent table.



Examples:

Get the nth maximum salary from the emp table.

Sol:

```
SELECT a.empid, a.ename, a.sal FROM EMP a  
WHERE 2=(SELECT count(distinct(b.sal)) FROM EMP b WHERE a.sal<b.sal)
```

Note: The above query is to calculate 3rd maximum salary. To calculate nth maximum salary specify the number “n-1” in place of “2”.

Get the Top 3 maximum salaries from emp table.

Sol:

```
SELECT a.empid,a.ename,a.sal FROM EMP a  
WHERE 3>(SELECT count(distinct(b.sal)) FROM EMP b WHERE a.sal<b.sal)  
Order by a.sal desc
```

Output:

Empid	Ename	Sal
104	Chandu	50000
102	Avinash	39000
103	Sumit	38000

Get the details of employees who are not the managers

Sol:

```
SELECT a.empid,a.ename FROM EMP a  
Where 0=(SELECT count(*) FROM EMP b WHERE a.empid=b.mgrid)
```

Output:

Empid	Ename
102	Avinash
106	vivek

Get the details of employees who are the managers for more than one employee

Sol:

```
SELECT a.empid,a.ename FROM EMP a  
Where 1<(SELECT count(*) FROM EMP b WHERE a.empid=b.mgrid)
```

Output:

Empid	Ename
103	Sumit



BUILT IN FUNCTIONS

SQL Server provides a lot of functions, which can be used as calculated fields as part of column lists in a SELECT statement. Such functions are called as Built-in Functions.

Types of Built-in Functions:

Based on the type of data we are using inside the functions, built-in functions are categorized into the following ways.

- 1) Aggregate Functions
- 2) Numeric Functions
- 3) Date and Time Functions
- 4) String Functions

1) Aggregate Functions:

Aggregate functions are used to produce summary data using tables.

Function	Parameters	Description
AVG	(ALL/DISTINCT] expression	Returns the average of values specified in the expression, either all records or distinct records
SUM	(ALL/DISTINCT] expression)	Returns the sum of values specified in the expression, either all records or distinct records.
MIN	(expression)	Returns the minimum of a value specified in the expression.
MAX	(expression)	Returns the maximum of a value specified in the expression.
COUNT	(ALL DISTINCT expression)	Returns the number of unique or all records specified in expression.

Examples of Aggregate functions

SELECT 'Average Price'=AVG(price) FROM titles	Returns the average value of all the price values in the titles table with user-defined heading.
SELECT 'Sum'=SUM(DISTINCT advance) FROM titles	Returns the sum value of all-the unique advance values in the titles table with user-defined heading.
SELECT 'Minimum Ytd Sales'=MIN(ytd_sales) FROM titles	Returns the minimum value of ytd_sales value in the titles table with user-defined heading.
SELECT 'Maximum Ytd Sales'=MAX(ytd_sales) FROM titles	Returns the maximum value of ytd_sales in the titles table with user-defined heading.



SELECT 'Unique Price'= COUNT(DISTINCT price) FROM titles	Returns the number of unique price values in the titles table with user-defined heading.
SELECT 'Price=COUNT(price) FROM titles	Returns the number of total number of price values in the titles with user-defined heading.

2) NUMERIC FUNCTIONS:

SQRT: Gets the square root of a number

EX: Select SQRT(25) --o/p: 5

SQUARE: Gets the square of a number

Ex: select SQUARE(4) --o/p: 16

POWER: Gets the power a number

Ex: select POWER(2,3) --o/p: 8

ROUND: Used to round a floating point number to the required no.of decimal places

Ex: 1. Select ROUND(12.347,2) --o/p: 12.35

CEILING: Returns the nearest higher integer value for the given number

Ex: select CEILING(12.345) --o/p:13

FLOOR: Returns the nearest smaller integer value for the given number

Ex: select FLOOR(12.345) --o/p:12

3) Date and Time Functions:

GETDATE: Returns system date and time

Ex: select GETDATE() --o/p: 2010-06-10 22:44:18.187

DATEPART: Returns the part of given date or time.

Syn: DATEPART(Format,Date)

Format	Output	Format	Output
dd	Days	mi	Minutes
mm	Months	ss	Seconds
yy	Years	ms	Milliseconds
hh	Hours	dw	Day of Week
mi	Minutes	dy	Day of Year
ss	Seconds		

ex: 1. select datepart(dd,getdate())--10



DATENAME: It Returns the name of the corresponding datepart. If the specified datepart is not having a name, this function will return that value directly.

Syn: DATENAME(datepart,date)

Ex:

```
1. select datename(dw,getdate()) --Friday
2. select datename(mm,getdate()) --June
3. select datename(dd,getdate()) --10
4. select datename(dw,'10/21/77') --Friday
5. calculate how many employees joined in each weekday from the emp table
select datename(dw,hiredate),count(*) from emp
group by datename(dw,hiredate)
```

DATEADD: It is used to add a number to a part of a date. If we specify a negative value, the number will be subtracted from the given datepart.

Syn: DATEADD(datepart,n,date)

Ex:

```
select dateadd(dd,3,getdate()) --2010-06-13
```

```
2. Getting the weekday name which was 10 days before today's date
sol:-select datename(dw,dateadd(dd,-10,getdate()))
--o/p: Saturday
```

DATEDIFF: Returns the difference between two given dates. The output will be the subtraction of date1 from date2. The difference can be in days or months or years etc., based on the interval specified.

Syn: DATEDIFF(interval, data1, date2)

Ex: Get the experience of each employee from the employee table.

```
ex: Get the experience of each employee from the employee table.
SQL: SELECT empid,ename,datediff(yy,hiredate,getdate()) FROM EMP
```

f. **CONVERT:** SQL server handles certain datatype conversion automatically. If a character expression is compared with an int expression, SQL server makes the conversion automatically for the comparison(implicit conversion).

The **CONVERT** function is used to change data from one type to another when SQL server cannot implicitly perform a conversion. Using the **CONVERT** function, data can be modified in variety of styles.

The syntax is:

```
CONVERT(datatype[(length), expression[,style]])
```



Ex:

```
SELECT Ytd_Sales=CONVERT(char(10),ytd_sales) FROM titles.
```

```
SELECT CONVERT(char,getdate(),101) --06/10/2010
```

4) STRING FUNCTIONS:

FUNCTION	DESCRIPTION
CHARINDEX('pattern', expression)	Returns the starting position of the specified pattern.
LOWER (character_expression)	Converts two string and evaluates the similarity between them on a scale of 1 to 4.
LTRIM(character_expression)	Returns the data without leading brackets
REPLICATE(char_expression, integer_expression)	Repeats a character expression a specified number of times.
REVERSE(character_expression)	Returns the reverse of character expression.
RIGHT(character_expression, integer_expression)	Returns the part of the character string from the right.
RTRIM(character_expression)	Returns the data without trailing blanks.
SPACE(numeric_expression)	Returns a string of repeated spaces. The number of spaces is equal to the integer expression.
UPPER(character_expression)	Converts the character expression into upper case.



OPERATORS

In order to improve the functionality of condition there by to retrieve user required information we can make use of special operators in where condition.

LIKE	UNION
IN	UNION ALL
ANY	INTERSECT
BETWEEN	EXCEPT

LIKE:

It is used to determine whether a character column is satisfying specific style or not. The style can be verified by using some special symbols called “wild-card characters”.

Syn: <EXP> [NOT] LIKE ‘pattern’

<u>Wild-card</u>	<u>Description</u>
%	Any no.of characters
_ (Underscore)	Any single character
[]	Any single character within range
[^]	Any single character not in range

Ex:

1) Select empid, ename from emp Where ename LIKE ‘s%’

→Returns employee details whose name starts with “S”.

2) Select empid, ename from emp Where ename LIKE ‘A_i%’

→Returns employee details whose name starts with “A” containing third character “i”.

3) Select * from emp where ename like ‘%[LH]’

→Returns details whose names end with L/H

IN:

It is used to verify whether a column is equal to any one of the list of values specified or not. The values can be specified directly or by using a SELECT statement.

Syn: <EXP > [NOT] IN (<List of values> | <SELECT stmt>)

Ex: 1) Select * from emp where sal IN(10000,20000,30000)

2) Get the departments which are not having emps.

Select dno, dname from dept

Where dno NOT IN (select distinct(dno) from emp)

3) Get the departments which are having emps.

Select dno, dname from dept

Where dno IN (select distinct(dno) from emp)



4). Get the details who are not managers.

Select * from emp

Where empid NOT IN (select mgrid from emp where mgrid is NOT NULL)

Note: While using NOT the values to be checked must not contain any NULL values.

ANY:

It is also used to compare whether a column value is existing in a list of values or not. Here we can use any kind of comparisons.

Syn: <exp> { < | > | !< | !> | <= | >= | = | != } ANY (<SELECT Stmt>)

Ex: Get the employees whose salary is greater than atleast one of employee.

Sol: select * from emp where sal > ANY (select sal from emp)

BETWEEN:

It is used to verify whether a column is existing in a range of values or not.

Syn: <EXP> BETWEEN <val1> AND <val2>

Ex: select * from emp where hiredate BETWEEN '1/1/92' AND '12/31/94'
(OR)

Select * from emp where datepart (YY,hiredate) BETWEEN 1992 AND 1994

UNION ALL:

It is used to get combining data b/w two tables when the both tables are having same structure.

Ex:

- 1) CREATE DATABASE DB3
- 2) GO
- 3) USE DB3
- 4) GO
- 5) CREATE TABLE PRODUCTS
- 6) (PROID INT, PRONAME CHAR(10), PROQTY INT)
- 7) GO
- 8) INSERT INTO PRODUCTS VALUES
- 9) (3,'p3',200),(4,'p4',500),(6,'p6',700)
- 10) GO
- 11) CREATE DATABASE DB2
- 12) GO
- 13) USE DB2
- 14) GO
- 15) CREATE TABLE PRODUCTS



```
16) (PROID INT,PRONAME CHAR(10),PROQTY INT)
17) GO
18) INSERT INTO PRODUCTS VALUES
19) (1,'p1',200),(2,'p2',500),(3,'p3',300)
20) GO
21) SELECT * FROM PRODUCTS
22) UNION ALL
23) SELECT * FROM DB3.DBO.PRODUCTS
```

UNION:

It is similar to the previous operator. But it will not display duplicate data.

INTERSECT:

It is used to get the common data b/w queries.

Ex: Get the departments which are having employees

```
Select * from dept where dno IN(select dno from dept INTERSECT select dno from emp)
```

EXCEPT:

It is used to get the data available in 1st query NOT in the 2nd query

Ex: Get the departments which are not having emps.

```
Select * from dept where dno IN(select dno from dept EXCEPT select dno from emp)
```



Stored Procedure

A SP is a database object that contains a set of pre-compiled re-executable statements as a unit.

The main advantages of a SP are:

- Reusability
- High performance

Reusability:

If we are executing some set of statements repeatedly instead of writing the code again & again. We can specify the code under the procedure and activate the procedure whenever it is required.

High performance:

A procedure will contain pre-compiled code inside it. Whenever we execute the procedure it directly executes the program without compilation. This will reduce execution time there by improves performance.

Syn:

```
1) CREATE PROCEDURE <Procedure Name>
2)    (@parameter1 <datatype> [output, @parameter2])
3) AS
4) BEGIN
5)    -----
6)    -----
7)    <Executable Stmt>
8)    -----
9) END
```

Parameters:

An argument or parameter will be used to control the execution of a procedure. SQL-Server supports 2 types of parameters. They are:

- I/P Parameters
- O/P Parameters

I/P Parameters:

These will be used to pass a value while calling the procedure. Based on this value the procedure execution will be done.



O/P Parameters:

These parameters will be used to assign a calculated value in the definition of the procedure. Before the procedure call O/P parameter doesn't contain any value. Once the procedure execution completes the O/P parameter will get some value.

Ex: Create a procedure to insert a record into products table.

```
1) Create procedure spaddprod(@pid int,@pn char(10),@pqty int)
2) AS
3) BEGIN
4) Insert into products values(@pid,@pn,@pqty)
5) END
```

Calling a procedure:

The "EXEC" statement can be used to call the procedure by passing values to the parameters.

Syn: EXEC <procedure name> <val1>,val2>,@parameter output

Ex:	EXEC spaddprod 1,'P1',200	→Correct
	EXEC spaddprod 1,'P1',200	→Runtime Error (Execute 2 nd time)

Exception Handling:

At the time of execution of program some times we might get errors because of wrong user input or incorrect programming language. It is the responsibility of developer to specify necessary steps to follow to continue the program execution. This process is called as error/exception handling.

To implement this we use TRY & CATCH blocks. In the TRY block we will specify executable statements and in CATCH block we will specify the error handling code.

Ex: Modify previous procedure by enabling exception handling.



TRIGGERS

A trigger is complex user defined integrity constraints that can be apply on the data base operations. Like a SP it is also one of the ways to implement logic in the data base. The differences between them are:

- 1). A procedure contains I/P & O/P parameters where as a trigger doesn't contain any parameters.
- 2). In order to execute the logic of procedure it must be called by user where as a trigger will be activated automatically by SQL-Server.

Execution Process:

- When we execute an INSERT command result of the command will be stored in the buffer with a table name called "inserted". When we execute a DELETE or UPDATE command then the old values will be stored in the buffer with a table name "deleted". These tables are called as magic/logical/working tables.
- After this task SQL-Server will search the table to identify the trigger corresponding to the operation. If it is found the logic of the trigger will be activated automatically.

Types of Triggers:

- Based on the time when the trigger is activating triggers are divided into following ways.
 - Before Triggers
(Instead of triggers)
 - After Triggers
 - DDL Triggers
 - DML Triggers
- Before triggers will be activated prior to the actual trigger execution. After triggers will be executed after performing an operation.
- DDL triggers will be developed for DDL operations such as CREATE db, ALTER db on server level and create table, alter table on d level.
- DML triggers will be developed on a table INSERT/DELETE/UPDATE operations.



User Defined Functions (UDF's)

An UDF is a db object that contains a set of pre-compiled re-executable sql statements as a unit. Like a SP it is also one of the way to implement the logic in db. The difference between them are:

- 1) A procedure contains I/P & O/P parameters where as an UDF contains only input parameters.
- 2) A procedure can't be called from a SELECT statement where as a function can be called anywhere from SQL-Server.

Syn:

```
1) CREATE FUNCTION<function name>
2) (@parameter1 <data type>,@parameter2..)
3) RETURNS <datatype>
4) AS
5) BEGIN
6) -----
7) -----
8) <executable stmt>
9) -----
10) RETURN(<Expression>)
11) END
```



CURSORS

A cursor is a pointer to the result of a SELECT statement. The following are the features provided by a cursor.

- Allowing positioning at specific rows of the result set.
- Retrieving one row or block of rows from the current position in the result set.
- Supporting data modifications to the rows at the current position in the result set.
- Supporting different levels of visibility to changes made by other users to the database data that is presented in the result set.
- Providing Transact-SQL statements in scripts, stored procedures, and triggers access to the data in a result set.

Types of Cursors:

- 1) Static cursors
- 2) Dynamic cursors
- 3) Forward-only cursors
- 4) Keyset-driven cursors

Static cursors detect few or no changes but consume relatively few resources while scrolling, although they store the entire cursor in tempdb. Dynamic cursors detect all changes but consume more resources while scrolling, although they make the lightest use of tempdb. Keyset-driven cursors lie in between, detecting most changes but at less expense than dynamic cursors.

Cursor Statements:

We can control the records of a result set with the use of these cursor statements.

They are:

- DECLARE
- OPEN
- FETCH
- CLOSE
- DEALLOCATE

DECLARE Statement is used to provide cursor declarations such as name
Of the cursor, the select statement to which result set the cursor should point etc.,



OPEN Statement starts using the cursor. It executes the select statement in the cursor and points the cursor to the first record of the result set.

FETCH statement is used to retrieve the data in the current location and navigate the cursor to the required position.

CLOSE statement stops using the cursor. But the resources used by the cursor are still open.

DEALLOCATE statements removes entire resources that are used by the cursor

@@FETCH STATUS: Returns the status of the last cursor **FETCH** statement issued against any cursor currently opened by the connection

Return value	Description
0	FETCH statement was successful.
-1	FETCH statement failed or the row was beyond the result set.
-2	Row fetched is missing.

Example:

Suppose there is a table with some data and indexes. After the index creation, if any modifications are performed on the table, those changes will not reflect directly into the index structure. We have to refresh the index structure on all table using the following statement.

```
DBCC DBREINDEX (<tablename>)
```

The above command should be executed for all tables of the database. To perform this task we can make use of the following cursor program:

```
declare @tblname varchar(255)

declare tblcursor cursor
for select table_name from information_schema.tables
where table_type='base table'

open tblcursor

fetch next from tblcursor into @tblname

while @@fetch_status=0
begin
    dbcc dbreindex(@tblname)
    fetch next from tblcursor into @tblname
end

close tblcursor
deallocate tblcursor
```



VIEWS

A view is a database object, which contains a query in it. Whenever we are executing a select statement repeatedly, instead of specifying the query again and again, we can store the query under an object and execute that object whenever the output is required. That object is called a view.

A view is also called as a virtual table, which gives access to a subset of columns from one or more tables. Hence, a view is an object that derives its data from one or more tables. These tables are referred to as the base tables or the underlying tables.

Views ensure security of data by restricting access to the following data:

- ⊗ Specific rows of the table
- ⊗ Specific columns of the table.
- ⊗ Rows fetched by using joins.
- ⊗ Statistical summary of data in a given table.
- ⊗ Subsets of another view or a subset of views and tables

Creating Views

Using the CREATE VIEW statement we can create a view.

Syntax: CREATE VIEW view_name
 [WITH ENCRYPTION]

The restrictions imposed on views are as follows:

- ⊗ A view can be created only in the current database.
- ⊗ A view can be created only if there is the SELECT permission on its base table.
- ⊗ The SELECT INTO statement cannot be used in a view declaration statement.
- ⊗ A view cannot derive its data from temporary tables.

Example:

Create view sampleview

As

Select sid, sname, s.cid, cname from student s inner join class c on s.cid=c.cid



The above statements create a view named sampleview, which contains the sid,sname and cid columns information from student table and cname column value from class table. The following statement can be used to execute a view:

```
Select * from sampleview
```

Altering a view

A view can be modified without dropping it. It ensures that the permissions on the view are not lost. A view can be altered without affecting its dependent objects, such as, triggers and stored procedures.

A view can be modified using the ALTER VIEW statement.

Syntax: ALTER VIEW view_name [WITH ENCRYPTION]
 AS
 Select_statement [WITH CHECK OPTION]

Example:
ALTER VIEW sampleview
AS
Select sid,sname,s.cid,cname,csize from student s inner join class c on s.cid=c.cid

Dropping a View

A view can be dropped from a database using the DROP VIEW statement. When a view is dropped, it has no effect on the underlying table(s). Dropping a view removes its definition and all permissions assigned to it. Further more if users query any views that refer to a dropped view, they receive an error message. However, dropping a table that refers to a view does not drop the view automatically. You must drop it explicitly.

The syntax of DROP VIEW statement is:

```
DROP VIEW view_name
```

Where view_name is the name of the view to be dropped.

It is possible to drop multiple views with a single DROP VIEW. A comma in the DROP statement separates each view that needs to be dropped.



Renaming a View

A view can be renamed with out dropping it. This also ensures that the permissions on the view are not lost.

The guidelines to be followed for renaming a view as follows.

- ⊗ The view must be in the current database.
- ⊗ The new name for the view must follow the rules for renaming identifiers.
- ⊗ Only the owner of the view and the database in which view is created can rename a view.

A view can be renamed by using the `sp_rename` system stored procedure.

The syntax of `sp_rename` is

Syntax: `Sp_rename old_viewname ,new_viewname`

Example: `sp_rename sampleView, studentinfoView`

NOTE: Generally it is possible to apply modifications to the view with the use of insert, delete and update commands. But all views are not updatable. If the modification to the view is affecting multiple base tables, then we can't modify the view. If the modification to the view is affecting a single base table, then we can perform that modification.

Ex:

Insert into sampleview values(111,'abc',222,'xyx')

--Error, since this modification affects both student and class tables.

Update sampleview set sid=99 where sid=1

--Correct, since this modification affects only student table.



SQL Server Security

SQL Server validates users at two levels of security on database user accounts and roles.

- 1) Login Authentication
- 2) Permissions Validation

The authentication stage identifies the user using a login account and verifies only the ability to connect with SQL Server. If the authentication is successful, the user is connected to SQL Server. The user then needs permissions to access database on the server, which is done by using an account in each database, mapped to the user login.

The permission validation stage controls the activities the user is allowed to perform in the SQL Server database.

Login Authentication:

A user must have a login account to connect to SQL Server. SQL Server provides two types of Login Authentications.

- 1) Windows NT Authentication
- 2) SQL Server Authentication

Windows NT Authentication

When using Windows NT Authentication, the user is not required to specify a login ID or password to connect to SQL Server. The user's access to SQL Server is controlled by the Windows NT account, which is authenticated when he/she logs on to the Windows Operating System.

SQL Server Authentication

When using SQL Server authentication, the users must supply the SQL Server login and password to connect to SQL Server. The users are identified in SQL Server by their SQL Server login.

Authentication Mode:

SQL Server can operate in two security modes:

Windows NT Authentication Mode:

Only Windows NT Authentication is allowed. Users cannot specify SQL Server Authentication.

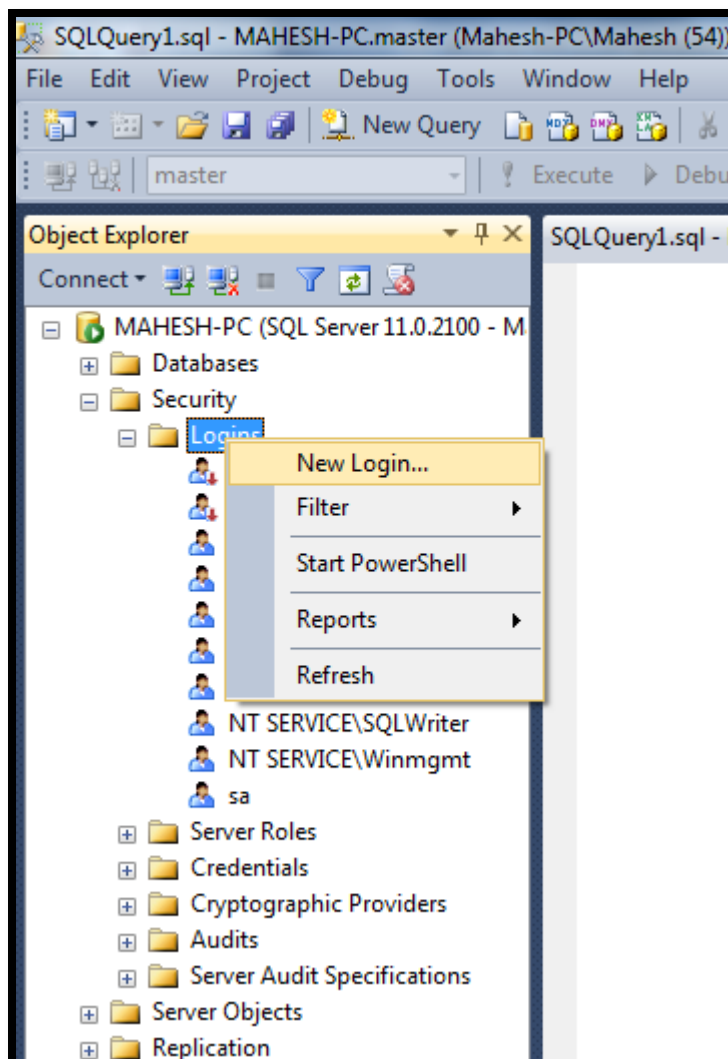
Mixed Mode:

It allows users to connect to SQL Server using Windows NT Authentication or SQL Server Authentication.

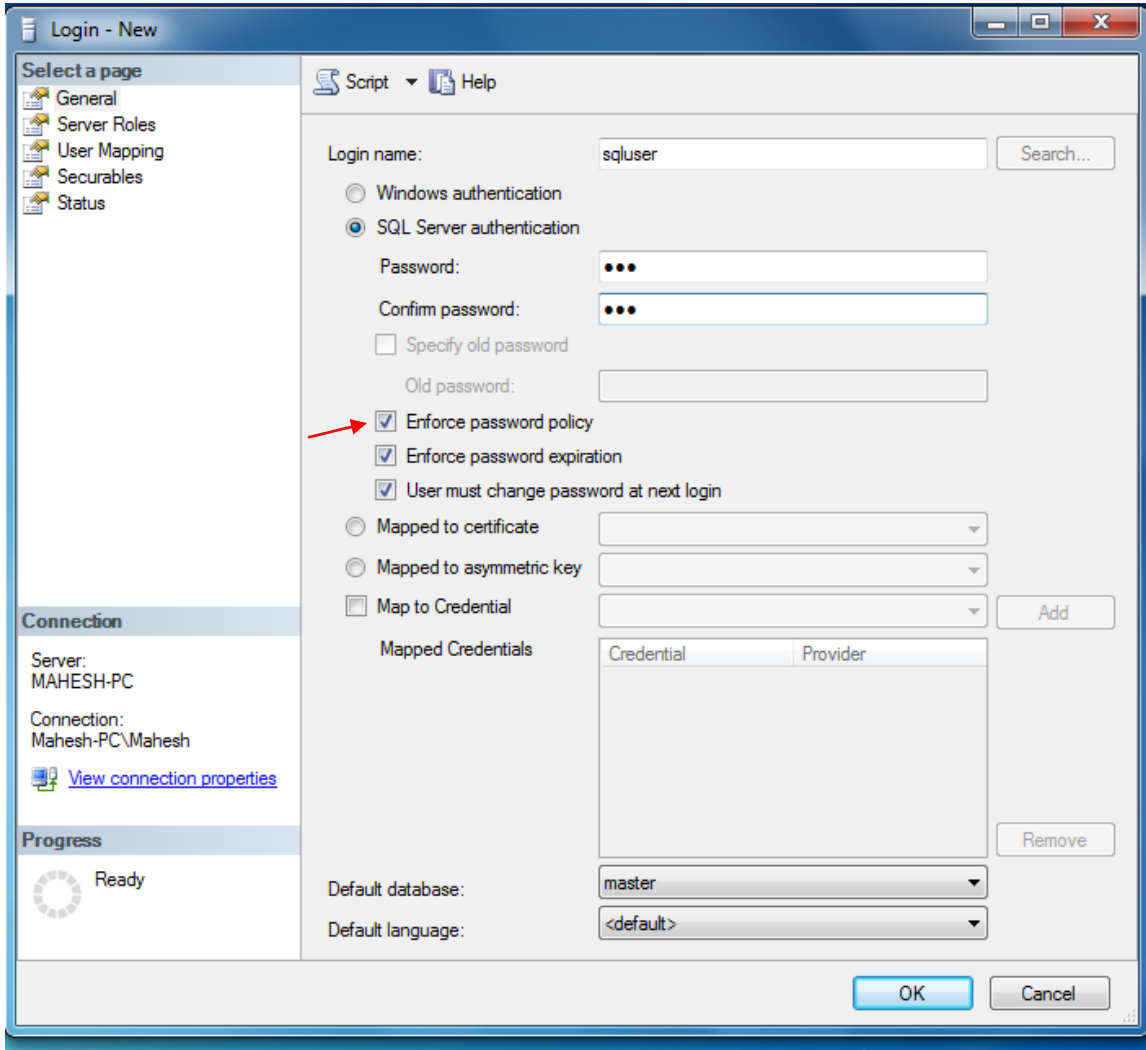


Create a User Login:

- ⊗ Expand the server
- ⊗ Expand security
- ⊗ Right click on “Logins” and select “New Login...”



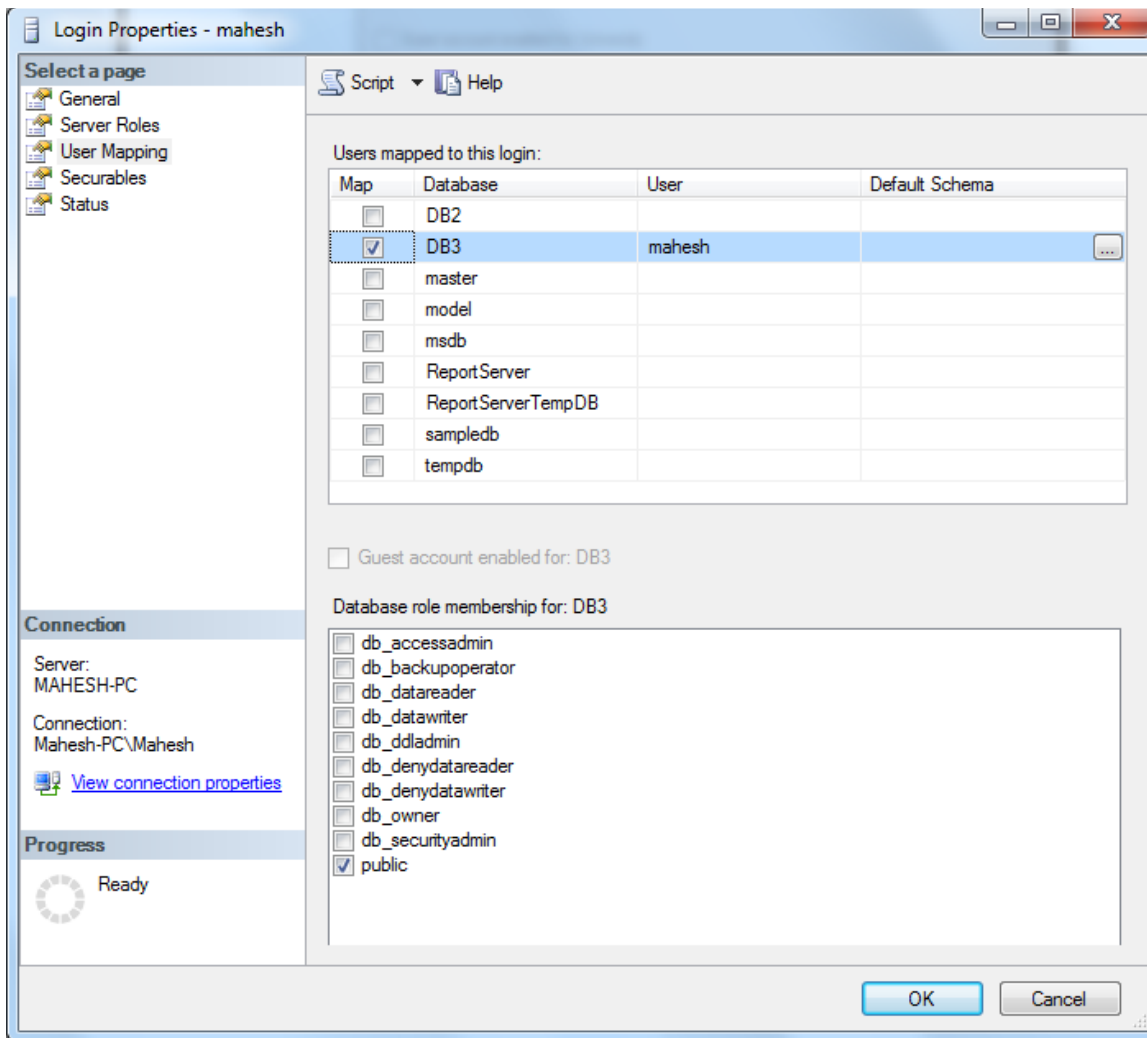
- In the window displayed enter a name to the user under *Name* Textbox
- Click the *SQL Server Authentication* option button to provide the password
- Enter “Password” and “Confirm Password”
- Uncheck “Enforce Password Policy” option.



Click "OK"

Providing access to a database for a user:

- ⊗ Right click on the login name and select properties
- ⊗ In the window displayed click on "User Mappings" tab
- ⊗ Make a check mark on the database for which the access has to be given.
- ⊗ Click "OK"



Granting Permissions on a table:

- ⊗ Right click on table name and select **"Properties"**
- ⊗ In the window displayed click on **"Permissions"**
- ⊗ In the window displayed click on **"Add"** button, **"Browse"** for the user, and make a check mark for the required user name and click **"ok"**



Select Users or Roles

Select these object types:
Users, Database roles, Application roles

Enter the object names to select (examples):

Check Names
Browse...

Browse for Objects

6 objects were found matching the types you selected.

Matching objects:

Name	Type
[dbUser]	User
[guest]	User
[public]	Database role
[sqlclass]	User
[sqluser]	User
[user1]	User

Column Permissions

Grant	With Grant	Deny
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Make a check mark for the desired operation.



Table Properties - PRODUCTS

Select a page

- General
- Permissions
- Change Tracking
- Storage
- Extended Properties


Script Help

Schema:

[View schema permissions](#)

Table name:

Users or roles:

Name	Type
 mahesh	User

Permissions for mahesh:

Explicit Effective

Permission	Grantor	Grant	With Grant	Deny
Control		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Delete		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Insert		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
References		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Select		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Take ownership		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Update		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

OK Cancel

Click "OK"



Normalization

It is the process of reducing redundancy & applying relation to the data.

It is a step-by-step process. Each step is called as a Normal Form. Each Normal Form will contain a set of rules that are to be satisfied by the tables.

In general 6 NF's are available. They are:

1NF, 2NF, 3NF, BCNF, 4NF, 5NF

Almost all real time project data base designs will be ended with 3NF's. They are:

1NF:

It defines that all the tables must not contain any duplicate data.

2NF:

It defines that all the non-key columns must depend on the primary key columns.



3NF:

It defines that all the non-key columns must depend only on the PK. There should be no internal dependency b/w the non-key columns.





Normalization

Chapter Objectives

- The purpose of normalization
- The Process of Normalization
- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

The Purpose of Normalization

Normalization is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise.

The process of normalization is a formal method that identifies relations based on their primary or candidate / foreign keys and the functional dependencies among their attributes.

The Process of Normalization

- Normalization is often executed as a series of steps. Each step corresponds to a specific normal form that has known properties.
- As normalization proceeds, the relations become progressively more restricted in format, and also less vulnerable to update anomalies.
- For the relational data model, it is important to recognize that it is only first normal form (1NF) that is critical in creating relations. All the subsequent normal forms are optional.

First Normal Form (1NF)

Unnormalized form (UNF)

A table that contains one or more repeating groups.

First Normal Form (1NF)

Repeating group = (propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

Unnormalized form (UNF)

A table that contains one or more repeating groups.

ClientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John kay	PG4	6 lawrence St, Glasgow	1-Jul-00	31-Aug-01	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-02	1-Sep-02	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 lawrence St, Glasgow	1-Sep-99	10-Jun-00	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-00	1-Dec-01	370	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-02	1-Aug-03	450	CO93	Tony Shaw

Figure 3 ClientRental unnormalized table



Definition of 1NF

First Normal Form is a relation in which the intersection of each row and column contains one and only one value.

There are two approaches to removing repeating groups from unnormalized tables:

1. Removes the repeating groups by entering appropriate data in the empty columns of rows containing the repeating data.
2. Removes the repeating group by placing the repeating data, along with a copy of the original key attribute(s), in a separate relation. A primary key is identified for the new relation.

1NF ClientRental relation with the first approach

The ClientRental relation is defined as follows,

ClientRental (clientNo, propertyNo, cName, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

ClientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 lawrence St, Glasgow	1-Jul-00	31-Aug-01	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-02	1-Sep-02	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 lawrence St, Glasgow	1-Sep-99	10-Jun-00	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-00	1-Dec-01	370	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-02	1-Aug-03	450	CO93	Tony Shaw

Figure 4 1NF ClientRental relation with the first approach



1NF ClientRental relation with the second approach

With the second approach, we remove the repeating group (property rented details) by placing the repeating data along with the original key attribute (clientNo) in a separate relation.

ClientNo	cName
CR76	John Kay
CR56	Aline Stewart

ClientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-00	31-Aug-01	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-02	1-Sep-02	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-99	10-Jun-00	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-00	1-Dec-01	370	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-02	1-Aug-03	450	CO93	Tony Shaw

Figure 5 1NF ClientRental relation with the second approach

Second Normal Form (2NF)

Second normal form (2NF) is a relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on the primary key.

The normalization of 1NF relations to 2NF involves the removal of partial dependencies. If a partial dependency exists, we remove the function dependent attributes from the relation by placing them in a new relation along with a copy of their determinant.



2NF ClientRental relation

The ClientRental relation has the following functional dependencies:

fd1	clientNo, propertyNo \rightarrow rentStart, rentFinish	(Primary Key)
fd2	clientNo \rightarrow cName	(Partial dependency)
fd3	propertyNo \rightarrow pAddress, rent, ownerNo, oName	(Partial dependency)
fd4	ownerNo \rightarrow oName	(Transitive Dependency)
fd5	clientNo, rentStart \rightarrow propertyNo, pAddress, rentFinish, rent, ownerNo, oName	(Candidate key)
fd6	propertyNo, rentStart \rightarrow clientNo, cName, rentFinish	(Candidate key)

2NF ClientRental relation

After removing the partial dependencies, the creation of the three new relations called Client, Rental, and PropertyOwner

Client

ClientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

ClientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-00	31-Aug-01
CR76	PG16	1-Sep-02	1-Sep-02
CR56	PG4	1-Sep-99	10-Jun-00
CR56	PG36	10-Oct-00	1-Dec-01
CR56	PG16	1-Nov-02	1-Aug-03

PropertyOwner

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	370	CO93	Tony Shaw

Figure 6 2NF ClientRental relation

Third Normal Form (3NF)

Transitive dependency

A condition where A, B, and C are attributes of a relation such that if $A \rightarrow B$ and $B \rightarrow C$, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).

Third normal form (3NF)

A relation that is in first and second normal form, and in which no non-primary-key attribute is **transitively** dependent on the primary key.

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies by placing the attribute(s) in a new relation along with a copy of the determinant.



3NF ClientRental relation

The functional dependencies for the Client, Rental and PropertyOwner relations are as follows:

Client

fd2 clientNo \rightarrow cName (Primary Key)

Rental

fd1 clientNo, propertyNo \rightarrow rentStart, rentFinish (Primary Key)

fd5 clientNo, rentStart \rightarrow propertyNo, rentFinish (Candidate key)

fd6 propertyNo, rentStart \rightarrow clientNo, rentFinish (Candidate key)

PropertyOwner

fd3 propertyNo \rightarrow pAddress, rent, ownerNo, oName (Primary Key)

fd4 ownerNo \rightarrow oName (Transitive Dependency)

3NF ClientRental relation

The resulting 3NF relations have the forms:

Client (clientNo, cName)
Rental (clientNo, propertyNo, rentStart, rentFinish)
PropertyOwner (propertyNo, pAddress, rent, ownerNo)
Owner (ownerNo, oName)

3NF ClientRental relation

Client

ClientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

ClientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-00	31-Aug-01
CR76	PG16	1-Sep-02	1-Sep-02
CR56	PG4	1-Sep-99	10-Jun-00
CR56	PG36	10-Oct-00	1-Dec-01
CR56	PG16	1-Nov-02	1-Aug-03

PropertyOwner

propertyNo	pAddress	rent	ownerNo
PG4	6 lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	370	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

Figure 7 2NF ClientRental relation