**1) SQL Server Overview**

2) **Analyzing physical structure of Sql Server** database in terms of Primary data files and Log Files

3**) Analyzing the different objects** and their relationships with each other which Sql Server uses to maintain the database like master database, system defined objects in it like procedures and users.

**4) Creating database** from query analyzer and from Enterprise Manager

**5) Basic Sql statements :-**

**a) CREATE TABLE** : used to create table

syntax=>. CREATE TABLE table\_name

( field1 datatype(size),

field2 datatype(size),

---

fieldn datatype(size)

)

e.g. CREATE TABLE employee

(

empcode int,

ename varchar(20),

salary int

)

**b) INSERT**:-use to add record in a table

**1)inserting in all columns:**

e.g. INSERT INTO employee values(1,’AAA’,100)

**2)inserting into fewer columns:**

e.g. INSERT INTO employee(empcode,salary) VALUES(1,1000)

**c) UPDATE** :- use to modify the existing record

**1)Updating all records**:

e.g. UPDATE employee

SET salary=salary+1000

**2)Updating fewer records:**

e.g. UPDATE employee

SET salary=salary + 1000

WHERE empcode=1001

**3)Updating more than one columns**:

e.g. UPDATE employee

SET salary=salary+1000, name=’newname’

WHERE empcode=1001

**d) SELECT :** used to query records:

**1)Displaying all records :**

e.g. SELECT \* FROM employee

**2)Displaying fewer columns:**

e.g. SELECT empcode,salary FROM employee

**3)Displaying filtered records:**

e.g. SELECT \* FROM employee

WHERE salary>1000

**4)Adding user-defined heading in the display:**

e.g. SELECT empcode “Employee Code”,ename

“Employee Name”,Salary

FROM employee

5**)Adding computed columns while displaying:**

e.g. SELECT empcode,ename,salary, salary\*0.1 “BONUS”

FROM employee

**6)Working with ALTER command:-** used to alter table structures and constraints

**a)Adding new column later:**

syntax=>

ALTER TABLE table\_name ADD newcolumn datatype(size)

e.g. ALTER TABLE employee ADD deptcode int

**b)Removing columns from table:**

syntax=>

ALTER TABLE table\_name DROP COLUMN column\_name

e.g. ALTE TABLE employee DROP COLUMN deptcode

**c) Modifying the structure of table:**

syntax=>

ALTER TABLE table\_name ALTER COLUMN column\_name

datatype(size)

e.g. ALTER TABLE employee ALTER COLUMN ename

VARCHAR(25)

**Note : we can modify only the size of columns and decreasing size is only possible if no truncation of data is happening. Also while doing any kind of changes to database whether to the structure of the object or data in the object it is only possible if no constraints are violated else it will be rejected even if it is syntactically correct.**

**d)Adding and removing constraints: will be there in next topics**

**e) Displaying the structure of a table or any object:**

syntax: sp\_help object\_name

e.g. sp\_help employee

**f) Adding user-defined datatype:**

syntax: sp\_addtype typename,’datatype’,’constraints’

e.g. sp\_addtype address ‘VARCHAR(25),’NULL’

Now this can be used in the database wherever required e.g.

CREATE TABLE tablename

(

field1 …..,

location address

)

**g) Generating script:-**

1)Importance:

2)Steps from enterprise manager

**7) Working with constraints:** used to maintain integrity of data inside database i.e. database must have correct and consistent data. This is done by putting rules on data through constraints

**Types of constraints:**

a)PRIMARY KEY

b)UNIQUE

c)DEFAULT

d)NULL

e)CHECK

f)FOREIGN KEY

**Note : constraints can have a single column or as a composite column where more than one columns**

**Also constraint can be applied at the time of creating table or later on using ALTER command**

1. **Adding constraints at the time of creating table:**

**i)PRIMARY KEY:- it maintains the uniqueness of record in a table like ‘empcode’**

CREATE TABLE employee

(

empcode int PRIMARY KEY,

ename varchar(25),

---

)

1. **UNIQUE:**- This also maintains the uniqueness of data but only for the column not for the record.

e.g. CREATE TABLE employe

(empcode int PRIMARY KEY,

regno int UNIQUE,

**Difference between PRIMARY KEY and UNIQUE constraints**

1. PRIMARY KEY =UNIQUE + NOT NULL i.e. Primary Key cannot contain NULL value but UNIQUE can contain.
2. There can be only one PRIMARY KEY in a table but can have multiple UNIQUE constraints.

**Syntax for composite column constraints:**

CREATE TABLE test

(

class int,

rollno int,

name char(20),

marks int

CONSTRAINT pky PRIMARY KEY (class,rollno)

)

**Syntax for UNIQUE constraint:-**

CREATE TABLE test

(

class int,

rollno int,

regno int UNIQUE,

name char(20),

marks int

)

1. **DEFAULT constraint:-** used to give some default value to the column to be entered automatically in case the column is not supplied by value.

**e.g.**

CREATE TABLE test

(

empcode int,

ename char(20),

salary int DEFAULT 5000,

gender char(1) DEFAULT ‘M’

)

1. **NULL constraint:-**  used to specify the NULL or NOT NULL constraints i.e. empty or non-empty column.

**e.**g.

CREATE TABLE test

(

empcode int PRIMARY KEY,

ename char(10) NOT NULL,

salary int DEFAULT 5000

)

1. **CHECK constraint:-** used to check the validity of the data on some condition

e.g.

CREATE TABLE test

(

empcode int,

ename char(25),

salary int CHECK (salary>=5000 AND salary<=10000)

)

**Note: CHECK constraint cannot involved any other column in its condition criteria apart from the current column on which it is used.**

1. **FOREIGN KEY constraint:-**  used to develop a referential integrity between two columns of **either same table or different table**

It has dependency rule between each other.

***Condition for Foreign Key constraint:***

a)First column which is also called master or parent must have PRIMARY KEY constraint on it.

b) Second column on which Foreign Key is applied must refer to the parent key

a)First table (Master table):

CREATE TABLE department

(

deptcode int PRIMARY KEY,

dname char(20),

dhead int,

location char(20)

)

b) Second table(Child table):

CREATE TABLE employee

(

empcode int,

ename char(20),

salary int,

deptcode int REFERENCES department(deptcode)

)

**a) Insert Rule :- We cannot insert a record in a child table(employee) whose corresponding key value is not present in parent table(department).**

**b) Delete Rule :- We cannot delete from master table (department) if some dependent child records are present in child table(employee). First we have to delete the child records and then delete the parent record**

1. **Adding constraints later on (after creating table):**

**a) PRIMARY KEY :-**

First we have to remove the NULL constraints from the column because it gets added default when table is created but primary key requires NOT NULL column.

ALTER TABLE employee ALTER COLUMN ecode int NOT NULL

After this we have to add the constraint PRIMARY KEY

ALTER TABLE employee ADD CONSTRAINT pky PRIMARY KEY (empcode)

**b) Adding CHECK constraint :-**

ALTER TABLE test ADD CONSTRAINT chk CHECK (salary>=5000

AND salary<=10000)

**c) Foreign Key:-**

ALTER TABLE emp ADD CONSTRAINT fky FOREIGN KEY

(deptcode) REFERENCES department(deptcode)

1. **Dropping constraint:-**

ALTER TABLE test DROP CONSTRAINT constraint\_name

**8) JOINS:-** is used to retrieve information which is scattered in different tables due to normalization process which has been done at the time of designing the tables to reduce redundant columns(repetitions of data)

**Types of JOINS:-**

**a)INNER JOINS :-** which is used to retrieve matching records

**b)OUTER JOINS:-** which is used to retrieve matching as well as un-matched records also.

**CROSS JOIN:-** which is a cross product or Cartesian product of records in which each and every record of one table with every record of second table.\

This join does not have any condition criteria. E.g if first table has **‘m’** records

And second table has **‘n’** records then CROSS-JOIN will result **mxn** records

**Types of INNER JOINS:-**

1. **Natural join:-** in which redundant column is also taken

SELECT \*

FROM emp

JOIN

department

ON department.deptcode=emp.deptcode

1. **Equi-join :-** in which condition matched is of equality condition. This is the special case of natural join**(operator is =) else it is called Non-equijoin**

Syntax=>

**a)SQL syntax=> in which where clause is used instead of keywords**

SELECT e.empcode,e.ename,e.salary,e.deptcode,d.dname,d.dhead,d.location

FROM emp e, department d

WHERE e.deptcode=d.deptcode

**Note: this is the standard syntax for all RDBMS**

**b) Keyword syntax=>**

SELECT empcode,ename,salary,emp.deptcode, dname, dhead, location

FROM emp

**JOIN**

department

**ON** department.deptcode=emp.deptcode

**Types of OUTER JOINS:-**

**a)LEFT OUTER JOIN:-** in which matched records as well as un-matched

records from the left table is given

Join

e.g.

SELECT \*

FROM department **LEFT OUTER JOIN** emp

ON department.deptcode=emp.deptcode

**b)RIGHT OUTER JOIN:-** in which matched records from both the tables and un-matched from right table is given

e.g.

SELECT \*

FROM emp **RIGHT OUTER JOIN** department

ON department.deptcode=emp.deptcode

**c)FULL OUTER JOIN:-** which displays matched records from both the tables as well as un-matched records also from both the tables

e.g.

SELECT \*

FROM department **FULL OUTER JOIN** emp

ON department.deptcode=emp.deptcode

**d)SELF JOIN:-** It is a special case of INNER JOIN but in this the table is joined with itself not with any other table. This is used when the whole information is lying inside the same table not outside the table.

e.g.

**9) SET Operations:- It is** used to join tables or relations horizontally. Note that in this case number of columns and their data types must be the same.

**a)UNION:-** used to combine all the records from one relation with another relation. By default if any duplicate records are there it will be not be repeated in the UNION result. To get that repetition in the result we use UNION ALL

**e.**g.

SELECT \* FROM passed\_stud

**UNION [ALL]**

SELECT \* FROM failed\_stud

**b)INTERSECT:-** This is used to find the common records from two relations i.e. those records which are present in both will be listed out

e.g.

SELECT \* FROM failed\_stud

WHERE **EXISTS** (SELECT \* FROM passed\_stud WHERE passed\_stud.rno=failed\_stud.rno)

**c)MINUS:-** used to retrieve records which are not are present in both the relation i.e. un-common records.

e.g.

SELECT \* FROM failed\_stud

WHERE **NOT EXISTS** (SELECT \* FROM passed\_stud WHERE passed\_stud.rno=failed\_stud.rno)

**10) Nested query :-** Also called sub-query i.e. query within a query. It is used when the matching condition value in the where clause is not known. To get that we have to take the help of again one more query and that becomes the child or nested or sub-query in order to get the condition value.

e.g. To get all the employee records who belong to the department of an employee whose deptcode matches with them

SELECT \* FROM emp

WHERE deptcode=(select deptcode

from emp where empcode=2)

**Nested query is of two types:-**

1. **Single valued :-** where result of the sub-query is only a single value
2. **Multiple valued :-** where result of the sub-query gives more than one values

So for single value the operators are :

>,<,>=,<=, = (for numeric) and (LIKE is for character)

for multi value the operators are :

IN, ALL

**a) ALL:-**

Using the > comparison operator as an example, >ALL means greater than every value--in other words, **greater than the maximum value.**

For example, >ALL (1, 2, 3) means greater than 3.

**b) ANY:-**

>ANY means greater than at least one value, that is, **greater than the minimum.** So >ANY (1, 2, 3) means greater than 1

**11) Views :-** Views are the logical grouping of some columns and restricted records from a single or multiple tables. It is used in some **security** purpose.

It is a compiled SQL code which just execute when we call it rather than compiling it again and again like normal SQL statements. So it **optimizes query** execution also.

It is an **object** stored with database.

It is also used to **simplify** some complex query in the sense it is created only once with the whatever complexity it has and gets stored for further use.

**Note: views does not hold the data but it has links to the base table**

**Types of views:-**

**1)Projection views:-** Also called simple view which hold columns and records only from single table

**2) Join Views:-** Also called complex views which holds columns and records from more than one tables e.g. if the query used for the view is a JOIN.

**Points to be remembered:-** DML operations(INSERT, DELETE,UPDATE) on views are allowed on views only if it is a simple view.

**Note:**-To insert into complex or join views we have to take the help of **INSTEAD OF Trigger**

**Syntax=>** **for simple view**

CREATE VIEW my\_view

AS

SELECT empcode, salary FROM emp

WHERE salary >ANY (SELECT salary FROM emp

WHERE empcode=2 OR empcode=4)

**Syntax => for complex view**

CREATE VIEW xyz

AS

SELECT e.empcode, e.ename, e.salary, e.deptcode, d.dname, d.dhead, d.location

FROM emp e, department d

WHERE e.deptcode=d.deptcode

**Removing views:- syntax=>**

DROP VIEW view\_name

e.g. DROP VIEW xyz

**Altering views:-** ALTER VIEW is used to modify the view. Just write the new SELECT query in the ALTER VIEW command to change the view content.

e.g. ALTER VIEW xyz

AS

SELECT ………