SQL Queries.

SELECT → used to select data from Database

1] Select all the records in "customers" table
⇒ select * from customers

2] Select cust_name & city columns from cust table
⇒ select cust_name, city from customers

SELECT DISTINCT → return only distinct values.

3] select only distinct values from "country" column in cust table.
⇒ select ~~DISIN~~ DISTINCT country from cust

WHERE → Filter Records

4] select all customers from country "Mexico" in cust table.
⇒ select * from cust where country = 'Mexico';

5] select all fields from "cust" where country = 'Germany' & city = 'Berlin'
⇒ select * from cust where country = 'Germany' and city = 'Berlin'

6] select all fields from "cust" where city = 'Berlin' or "Munchen"
⇒ select * from cust where city = 'Berlin' or city = 'Munchen'

7] Select all fields from 'cust' where country is not Germany & USA:
⇒ select * from cust where not country = 'Ger' and not cnt = 'USA'

ORER BY → sort the result in ascendingor descending

8] select all custs from 'cust' sorted by "country" column.
⇒ select * from cust Order By country; // DESC

9] select * from cust order By country, custName.
// It orders by country if some rows have the same ~~name~~ country, it orders them by custName.

1] Where 4] NULL 7] SELECT 10] TOP
2] Order By 5] Insert into 8] Alter 11] MIN, MAX, COUNT
3] Having 6] UPDATE 9] DELETE 12] Avg, SUM

10] Insert into → insert new records in Table
:- Insert into cust (custName, city, country)
values ('Ajay', 'Pune', 'India')

NULL → field with no value

11] List all custs with NULL value in "Address" col.
:- select * from cust where Address IS NULL

12] List all custs with a value in "Address" col.
:- select * from cust where Address IS NOT NULL

UPDATE → modify the existing records in a table

13] Update the first custome with new person & city
:- update cust set custName = 'Ajay', city = 'Pune'
where custID = 1;

14] delete customer 'Ajay' from 'cust' table
:- delete from cust where custName = 'Ajay';

delete → delete existing records in a table

15] select first three records from "cust" table:
:- select TOP 3 * from cust;

Select TOP → To specify the no. of records to return

16] select first 50% of the records from "cust"
:- select TOP 50 PERCENT * from cust;

17] find the price of the cheapest product
:- select MIN(price) as smallestPrice from products;

MIN() → smallest value

18] find the price of the most expensive product.
:- select MAX(price) as LargestPrice from products;

MAX() → largest value

13) LIKE 16) Aliases 19) group by 22) Any / All
14) IN 17) JOIN 20) Having 23) select into
15) Between 18) Union 21) Exists 24) insert into

Page No.
Date

// NULL values are not counted

19) find the number of products
→ select COUNT(ID) from products
COUNT() → no. of rows that matches specified criteria.

20) find the average price of all products
→ select AVG(price) from products
AVG() → average value of numeric column.

21) find the sum of the 'quantity' fields in the products.
→ select sum(quantity) from products
SUM() → total sum of a numeric column.

22) select all customers with custName starting with "a".
→ select * from cust where custName like 'a%';
LIKE() → search for a specified pattern in column.

23) ending with 'a'.
→ select * from cust where custName like '%a';
wildcard % → zero, one, or multiple char

24) selects all customes with custName that have "or" in any position.
→ select * from cust where custName like '%or%';
wildcard → substitute one or more chars in string.

25) select all customers with custName that have 'r' in 2nd posn.
→ select * from cust where custName like '_r%';
wildcard _ → one, single char.

26) custName that starts with "a" and are at least 3 char in length.
→ select * from cust where custName like 'a__%';

27) custName that starts with 'a' and ends with 'o'.
→ select * from cust where custName like 'a%o';

28) custName that does not start with 'a'.
→ select * from cust where custName not like 'a%';

25) case

29) Select all customers with a city starting with any char, 'ondon'.
→ select * from cust where city like '_ondon'.

30) city starting with 'L', followed by any char, followed by 'n', followed by any char, followed by 'on'.
→ select * from cust where city like 'L_n_on'.

31) select all customers with city starting b,s,p.
→ select * from cust where city like '[bsp]%';

32) select all customers with city starting with a,b,c.
→ select * from cust where city like '[a-c]%';

33) with city NOT starting with b,s,p.
→ select * from cust where city like '[!bsp]%' ;
NOT like '[bsp]%'

34) Select customers that are located in 'Ger', 'UK', 'France'.
→ select * from cust where city in ('Ger', 'UK', 'France')
IN → allows you to specify multiple values in Where clause.

35) Oolta
→ select * from cust where city NOT in ('Ger', 'UK', 'France')
IN → shorthand for multiple OR cond^s.

36) Select customers that are from the same countries as suppliers.
→ select * from cust where country in (select country from suppliers)

37) Select all products with price b/w 10 & 20.
→ select * from products where price between 10 and 20
Between → selects values within given range.

38) Oolta
→ select * from prods where price not between 10 and 20
Inclusive : begin & end values [ ]

39) Select products with price blw 10 & 20, do not show products with ID 1,2,3
→ select * from prod where price between 10 and 20
and ID Not in (1,2,3)

40) select all orders with an date between 01-07-1996 & 31-07-1996.
→ select * from orders where date between '1996-07-01' and '1996-07-31'

41) → Select CustID as ID, CustName as Name from customers;
Aliases → Temporary Name for column.

42) → Select custID as ID, ContactName as [Contact person] from cust;
Exists for the duration of that query

43) Select all orders from customer with ID=4 (Around the Horn).
use the 'customers' and 'orders' tables, give them aliases of c & o
→ select o.ID, o.orderDate, c.CustName
from customers as c, orders as o
where c.CustName = 'Around the Horn' and
c.ID = o.ID;
To make column names more readable

44) select orders.ID, customers.Name, Orders.Date
from Orders
INNER JOIN customers ON orders.ID = Customers.ID
JOIN → used to combine rows from two or more tables

45) select orders.ID, customers.Name, Shippers.ShipperName
from Orders
INNER JOIN Customers ON orders.ID = customers.ID
INNER JOIN Shippers on orders.ShipperID = Shippers.ID
INNER JOIN → selects records that have matching values

46) select C.Name, O.ID from C
LeftJoN O
ON C.ID = O.ID
ORDER BY C.Name

47) select O.ID, E.Name from O
RIGHT JOIN E
ON O.ID = E.ID
Ordering By O.ID

LEFT JOIN →
L + L∩R

RIGHT JOIN →
R + L∩R

48) Select C.Name, O.ID
from C FULL OUTER JOIN O
ON C.ID = O.ID
Order By C.Name
FULL OUTER → LIUL2

49) select A.CName, B.CName, A.City
from Cust A, Cust B
where A.ID <> B.ID
AND A.city = B.city
ORDER by A.city
SELF JOIN → table join with itself

50) select city from C
union All → duplicate ✓
select city from S
order by city;
UNION → combine result set
of two or more select statement
UNION All → allow duplicates

51) select city, country from C
where country = 'Ger'
union All → duplicate allowed
select city, country from S
where country = 'Ger'
order by city.

52) List the no of customers in each country.
→ select count(ID), country from cust
group by country;
Group By → groups rows that have same values

53) List the no of customers in each country, sorted high to low.
→ select count(ID), country from cust
group by country
order by count(ID) DESC
uses aggregate functions to group the result-set by one or more

54) List the no of orders sent by each shipper.
→ select shippers.Name, count(Orders.ID) as NoofOrders
from orders left join shippers
ON orders.ID = shippers.ID
group by Name;

55) list the number of custome in each country, with more than 5 customer
→ select count(ID), country from customer
group by country having count(ID) > 5 ;

56) List & sort descending
→ select count(ID), country from customer
group by country having count(ID) > 5 order by count(ID) desc;

57) list the employees that have registered more than 10 orders.
→ select employee.Name, count(Orders.ID) as numofOrders
from Orders INNER JOIN Employees
ON Orders.ID = Employees.ID
Group by Name having count(orders.ID) > 10 ;
Having → because where keyword can't be used with aggregate f^n -

58) list if employee 'Ajay' or 'shyam' have register more than 25 orders.
→ select employee.Name, count(Orders.ID) as no.ofOrders
from orders INNER JOIN Employees
on Orders.ID = Employees.ID
where Name = 'Ajay' or Name = 'shyam'
group by name having count(Orders.ID) > 25 ;

59) list the suppliers with product price less than 20.
→ select supplierName from Suppliers
where exists (select productName from products where
prod.ID = supp.ID and price < 20)
Exists → To test for the existence of any record in subquery

60) List the suppliers with product price = 22
select suppName from suppliers
where exists (select prodName from products where
prod.ID = supp.ID and price = 22
→ returns true if subquery returns one or more records

61) List the ProductName if it finds any records in orders has quantity = 10.

→ select prodName from products
where productID = ANY (select ID from Orders where Q = 10)

ANY or ALL → to perform comparison b/w single column value & range of other values.

62) List prod-Name if it finds any records in orders has Q > 99.

→ select prod-Name from products
where prodID = Any (select ID from orders where Q > 99)

63) create backup copy of customers.

→ select * into backup from customers
select * into backup IN 'backup.mdb' from customers;

64) SELECT INTO → copies data from one table → another.

64) copy only few cols in new table.

→ select name, ID into backup from customers

65) copy only german customers into new table.

→ select * into backup from customers where country = 'Ger';

66) Copy data from more than one table into a new table.

→ select customers.Name, Orders.ID into backup
from customers left join orders
on orders.ID = customers.ID

67) create new, empty table using the schema of another.

select * into backup from table
where 1 = 0;

1) Create DB 4) backup db 7) Alter Tb
2) Drop DB 5) Create Tb 8) constraints
3) Show DB 6) Drop Tb 9) Index

Page No.
Date

68) copy all columns from one table to another.
→ insert into table2 insert into table2 (col1, col2, col3, ...)
select * from table1 select col1, col2, col3 from table1
where condition.; where condition;
INSERT INTO SELECT → copies data from one table & another.

69) copy only german suppliers into customers.
→ insert into customers (Name, city, country)
select name, city, country from suppliers
where country = 'Ger';
requires datatypes in source & target tables match..

70) select ID, quantity, ~~Text~~
case
when q > 30 then 'q > 30'
when q = 30 then 'q = 30'
else 'q < 30'
END as quantitytext
from orders.
Case → goes through cond^n & return when 1st cond^n met.

71) select Name, city, country
from customers
order by
(case
when CITY is ~~Not~~ null
then country
else CITY
END;

71) return an alternative value, when an expression is null
→ select prod, unitprice * (unitsinstock + ISNULL(UnitsOrder, 0))
from products
COALESCE(Units, 0)
ISNULL(x, 0) → return an alternative value if expre is NULL.

72) Create a DB "testDB" :- create database testDB
CREATE DB → Create new SQL DB

73) drop an existing DB :- Drop database testDB

74) Show DB :- show Databases

75) Create backup of existing db "testDB" to disk.
→ Backup database testDB
To DISK = 'D:\backups\testDB.bak';

76) Create table "Person" having, PID, LN, FN, Address & City.
→ create table person (PID int, LN varchar(10), FN0 varchar(20), Address varchar(20), City varchar(20))

77) Create a table "Test" copy of "cust" table.
→ create table Test as select custName, contName from cust

78) drop existing table "test" :- drop table test;

79) delete data inside a table, not the table itself
→ Truncate table test

80) Add "email" column to the "cust" table.
→ Alter table cust ADD email varchar(20);
Alter → used to add, delete, or modify column in existing table

81) Delete "email" column from "cust" table.
→ Alter table cust DROP column email;
To drop & add various constraints

82) Change data type of column
→ Alter table cust Alter column DOB date;

83) Rename a column in table cust
→ Alter table cust Rename column DOB to Date of Birth;

84) Create NOT NULL constraint on Age column.
→ Alter table Persons Alter column Age int NOT NULL;
CONSTRAINT → to specify rules for data in table.

85) Unique constraint :-
create table persons (ID int NOT NULL Unique, LN varchar(20) NOT NULL, FN varchar(20); Age int);
NOT NULL → enforces column to NOT accept NULL values

86) Two or more columns :-
→ create table persons (ID int NOT NULL, ~~Uniqu~~ LN varchar(20),
FN varchar(20), Age int, constraint UC_Person Unique (ID,LN)
UNIQUE → ensures that all values are different in column

87) create unique constraint on 'ID' column when table is created
→ Alter table person, add unique (ID);
More UNIQUE KEYS.

88) Two or more columns :—
→ Alter table person add constraint UC_Person Unique (ID,LN)

89) Drop unique constraint
→ Alter table persons drop constraint UC_Person;

90) Create Primary key on 'ID' column.
→ create table persons (ID int NOT NULL, Primary key,
LastName varchar(20) NOT NULL, FN varchar(20), Age int);
Primary key → uniques identifies each record in table.

91) To allow naming of PK constraint, for defining PK constraint on many columns. →
→ create table Persons (ID int, LN varchar(20), FN varchar(2
Age int, constraint PK_Person Primary key (ID, LN));
can't contain NULL values & contain UNIQUE.

92) Create PK constraint on ID column.
→ Alter table Persons ADD Primary key (ID);
It can be comb^n of (one, more fields) Only one PK.

93) ~~To allow~~ Two or more columns :-
→ Alter table Persons ADD constraint PK_Person PK (ID, LN);

94) Drop PK constraint → Alter table persons
drop constraint PK_Person;

95) Create Foreign key on 'PID' column
→ create table orders (OID int Primary key,
ONo int NOT NULL, PID int foreign key References Per(ID));
Foreign key → used to prevent actions that would destroy

96) Create FK on 'PID' column, Orders table is already created.
→ Alter table Orders Add Foreign Key (PID) References Per(ID));
rel^n b/w two tables.

95) → create table orders (OID int, ONo int NOT NULL,
PID int, PK (OID), constraint FK_Porder FK (PID)
References Persons (PID));
FK → a field or coll^n of fields in one table refers PK in another

96) → Alter table orders ADD constraint FK_Porder
FK (PID) References Persons (PID)
FK → child Table, PK → Parent Table.

97) DROP FK constrain → Alter table orders drop constraint FK_Porder

98) Create constraint on 'Age' column when 'Persons' table created.
check constraint → age of person > 18
→ create table persons (ID int NOT NULL, LN Varchar(25),
FN varchar(25), Age int, check (Age >= 18));
CHECK → used to limit the value range that can be placed in column.

99) two or more columns
→ create table persons (ID int, LN varchar(25), FN varchar(25)
Age int, city varchar(25), constraint PK_Person PK (ID),
constraint CHK_Person (Age > 18 and city = 'India'));

100) Create constraint on 'Age' column when table already created.
→ Alter table persons Add (check >= 18);

101) two or more columns
→ Alter table persons Add constraint CHK_Person check
(Age > 18 and city = 'India');

102) Drop check constraint → Drop constraint CHK_Person.
DEFAULT → used to set default value for column.

103) Set default value for city column when person table created.
→ create table persons (ID int NOT NULL, LN varchar(25), FN VC(25), Age int, city varchar(25) default 'Pune');
oDate date → Default GetDate());

104) Create ~~to~~ default when table already created.
→ Alter table persons ~~Alter city set default 'Pune'~~
add constraint Df_City Default 'Pune' for city.

105) Drop default constraint → Alter table persons
Alter column city Drop default

106) Create an index on LastName column in Person table.
→ create index idx_LN ON persons (LastName);
CREATE INDEX → retrieve data from DB more quickly.

107) two or more columns combo
→ create index idx_PN ON persons (firstName, LastName);
Updating table with indexes takes more time.

108) Drop index in ~~table~~ → Drop index tb_name, idx_name;

109) Defind PID column to be an auto increment PK field in persons
→ create table Persons (PID int identity(1,1) Primary key,
LN varchar(25) NOT NULL, FN varchar(25), Age int);
AUTO INCREMENT → it allows a unique no. to be generated
110) automatically when new record is inserted into a table.

Group by

group by clause is used to group a selected set of rows into a set of summary rows by the values of one or more columns or expressions.

It is always used in conjuction with one or more aggregate function → MAX, MIN, SUM, COUNT

```
select city, sum(salary) as Totalsalary
from tblEmployee group by city
```

```
select city, Gender, sum(salary) as TotalSalary, count(Gender) as TotalEmployees
from tblEmployee
group by city, Gender
order by city
```

Filtering Groups

Where clause is used to filter rows before aggregation.

HAVing clause is used to filter groups after aggregation.

```
Select city, sum(sal) as Total
from tblEmployee
where city='Mumbai'
group by city
```

```
select city, Sum(sal) as Total
from tblEmployee
group by city
having city='Mumbai'
```

Page No.
Date

Difference Where & Having

Where	Having
1) Select, Insert, Update	1) Only Select
2) Filter rows before aggregation	2) Filter groups, after aggregation.
3) aggregate functions can't be used in WHERE clause, unless it is in a subquery contained HAVING clause.	3) aggregate functions can be used in HAVING clause

```
select * from tblEmployee
 where sum(salary) > 4000
```

Can't be used
error will generate

```
select gender, city, sum(salary) as TotalSalary,
from tblEmployee                  COUNT(ID) as TotalEmp
group by Gender, City
having sum(salary) > 5000
```

Can be used

Joins in SQL Server

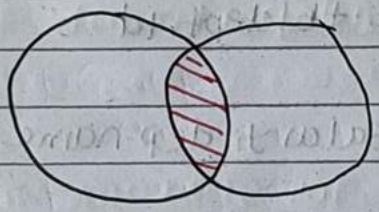
Used to retrieve data from two or more related tables. In general, tables are related to each other using FK constraints.

In sqlserver, there are different types of Joins

1. Inner Join
2. Outer Join → Left Join, Right Join, Full Join
3. Cross Join

Inner Join or Join

returns only the matching rows b/w both tables. Non-matching elements are eliminated.

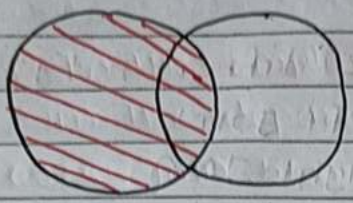


matching rows only
non-matching rows eliminated

1] Select name, gender, Salary, Dep-Name
from tblemployee
INNER JOIN tbldepartment
ON tblemployee.Dep-Id = tbldept.Id

2] Select name, gender, Salary, Dep-Name
from tblemp
JOIN tbldept
ON tblemp.dep-Id = tbldept.Id

Left outer Join or Left Join
returns all the matching rows + non-matching rows from the left table

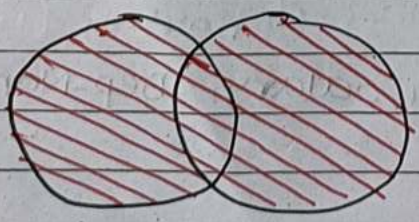


matching rows + non matching rows from the left table

1) select name, gender, salary, dep-name
from tblemp
LEFT OUTER JOIN tbldept
ON tblemp.deptId = tbldept.Id

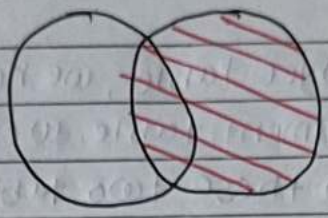
2) select name, gender, salary, dep-name
from tblemp
LEFT JOIN tbldept
ON tblemp.deptId = tbldept.Id

FULL OUTER JOIN or Outer JOIN
returns all rows from both left and right tables, including non-matching rows.



matching rows + non matching rows from both the table

Right Outer Join or Right Join
returns all the matching rows + non-matching rows from the right table



matching rows + non-matching rows from the right table

1) Select name, gender, salary, dep-name
from tbl-emp
RIGHT OUTER JOIN tbl-dept
ON tbl-emp.deptId = tbl-dept.Id

2) select name, gender, salary, dep-name
from tbl-emp
RIGHT JOIN tbl-dept
ON tbl-emp.deptId = tbl-dept.Id

---

1) Select name, gender, salary, dept-name
from tbl-emp
FULL OUTER JOIN tbl-dept
ON tbl-emp.deptId = tbl-dept.Id

2) Select name, gender, salary, dept-name
from tbl-emp
FULL JOIN tbl-dept
ON tbl-emp.deptId = tbl-dept.Id

CROSS JOIN

produces the cartesian product of the 2 tables involved in the join.

For ex:, In the Employee table, we have 10 rows and in the department table, we have 4 rows. so, a cross join b/w these two tables produces 40 rows

NOTE: CROSS JOIN should not have ON clause

CROSS JOIN :-

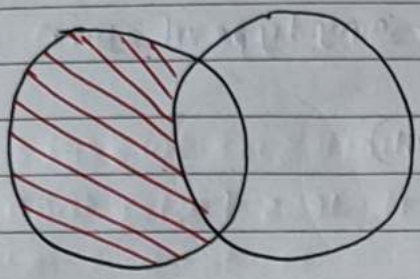
```
select name, gender, salary, dep-name
from tblemployee
CROSS JOIN tbldepartment
```

GENERAL Formula for JOINS :-

```
SELECT COLUMN-LIST
FROM   LEFT_TABLE-NAME
JOIN-TYPE RIGH-TABLE-NAME
ON     JOIN-CONDITION
```

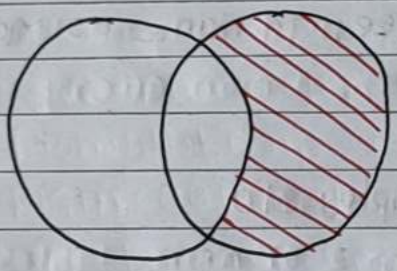
Advanced Joins

(1)



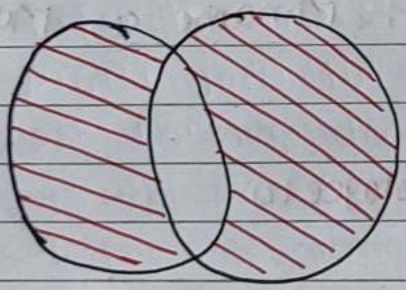
```
select name, gen, sal, Dep-name
from tbl-emp
LEFT JOIN tbl-dept
ON tbl-emp.deptId = tbl-dept.Id
WHERE tbl-emp.deptId IS NULL
```

(2)



```
select name, gen, sal, Dep-name
from tbl-emp
RIGHT JOIN tbl-dept
ON tbl-emp.deptId = tbl-dept.Id
WHERE tbl-dept.Id IS NULL
```

(3)



```
select name, gen, sal, Dep-name
from tbl-emp
FULL JOIN tbl-dept
ON tbl-emp.deptId = tbl-dept.Id
WHERE tbl-emp.deptId IS NULL
OR tbl-dept.Id IS NULL
```

Self Join

Joining a table with itself is called as self join
self join is not a different type of join
It can be classified under any type of join

1. Inner
2. OUTER (left, right, full)
3. cross

Left Outer Self Join

```
select E.name as Employee, M.Name as Manager
from tblEmp E
LEFT OUTER JOIN tblEmp M
ON E.ManagerID = M.EmployeeID
```

Inner self Join

```
select E.name as Employee, M.name as Manager
from tblEmp E
inner join tblEmp M
ON E.ManagerID = M.EmployeeID
```

Cross Self Join

```
Select E.name as Employee, M.name as Managers
from tblEmp E
cross join tblEmp M
```

Replacing NULL values

Three ways :-

1. ISNULL() function
2. CASE statement
3. COALESCE() function

(1)
```
select E.name as Emp, ISNULL(M.name,'No Mgr') as Mgr
from tblEmp E
LEFT JOIN tblEmp M
ON E.managerID = M.managerID
```

(2)
```
select E.name as Emp,
    CASE
        When M.name ISNULL Then 'No Mgr'
        Else M.Name
    END as Manager
from tblEmp E
LEFT JOIN tblEmp M ON E.mgrID = M.mgrID
```

(3)
```
select E.name as Emp, COALESCE(M.name,'No mgr') as Mgr
from tblEmp E
LEFT JOIN tblEmp M
ON E.mgrID = M.mgrID
```

Union and Union All and Joins

Difference b/w UNION and UNION ALL

1. Union removes duplicate rows, where as Union all does not.
2. Union has to perform distinct sort to remove duplicates, which make it less faster than Union all.

Sorting Results of a Union or Union All

Ordered by clause should be used only on the last SELECT statement in the Union query.

Difference b/w UNION and JOINS

Union combines rows from 2 or more tables, where JOIN combines columns from 2 or more tables

Stored Procedures

A stored procedure is a group of Transact SQL statements. If you have a situation where you write the same query again and again, you can save that specific query as a stored procedure and call it just by its name.

```
CREATE PROCEDURE spGetEmployees
AS
    BEGIN
        select Name, Gender from tblEmp
    END
```

To run stored procedure :-

```
spGetEmployees
Exec spGetEmployees
Execute spGetEmployees
```

Stored Procedure with Parameters

```
Create proc spgetEmployeesByGenderAndDept
@Gender nvarchar(20),
@DeptId int    WITH ENCRYPTION
AS
    BEGIN
        Select Name, Gender, DeptId from tblEmp
        Where Gender = @Gender and DeptId = @DeptId
    END
```

we cannot open or read the data whatever inside the SP or we can't modify

To run stored procedure with parameters :-

```
spgetEmployeesGenderAndDept @DeptId = 1, @Gender = 'M'
                    OR              'male', 1
```

To view the text, of the stored procedure
sp_helptext 'SPName'

To change the SP
ALTER PROCEDURE SPName ~~NewName~~

To delete the SP
DROP PROCEDURE 'SPName'
DROP PROC 'SPName'

To change the procedure, use ALTER PROCEDURE statement.

To delete the SP, use DROP PROCEDURE 'SPName'
DROP PROC 'SPName'

To encrypt the text of SP, use WITH ENCRYPTION
It is not possible to view the text of an encrypted SP.

To create a SP with OUTPUT Parameter, we use OUT or OUTPUT keywords

```
create proc getEmp
@Gender varchar(20),
@countofGender INT output
as
begin
    select @countofGender = count(*) from emp_table
    where @Gender = @Gender
end
```

To execute SP with out put parameters

```
declare @empcount int
exec getEmp 'male', @empcount OUTPUT
print @empcount
```

If you don't specify the output keyword, when executing the SP, the @empcount variable will be NULL

Useful stored Procedures.

System

1) sp-help proc-name :- view the information about the stored procedure like parameters, their datatypes

2) sp-helptext proc-name :- view the text of the stored procedure

3) sp-depends proc-name :- view the dependencies of the stored procedure.

- SP with return values

  whenever you execute the SP, it returns an integer value / status variable. Usually zero, indicates success others indicates failure.

CODE

```
create proc getcount
@count int ouptut
as begin
   select @count = count(ID)
   from tblemp
end
```

```
create proc getcount1
as begin
return (select count(ID) from emp)
end
```

EXECUTE

```
Declare @ans int
exec getcount 1, @ans out
print @ans
```

```
Declare @ans1 int
exec @ans1 = getcount1
select @ans1
```

In this case, Return wala case not applicable.

```
create proc getName
@id int,
@name varchar(20) output
as begin
  select @name = name from
  tbl where @id = @ID
```

```
create proc getName1
@ID int
as begin
return (select name from
tbl where ID = @ID)
end
```

```
Declare @ans varchar(20)
exec getName 1, @ans OUT
print @ans
```

```
declare @ans1 varchar(20)
exec @ans1 = getName1 1
print @ans1
```

Executing getname1 returns an error stating, 'conversion failed when converting the varchar value 'sam' to int'

Advantages of Stored Procedures

1. Execution Plan retention
2. Reusability
3. Reduces network traffic
4. Better security
5. Maintainability

SQL describes how a collection of data is to be organized.

Page No.
Date

What is SQL?

Structured query language is a computer language that we use to interact with a relational DB.

SQL is a tool for organizing, managing and retrieving archived data from computer DB.

Features of SQL

SQL is a non-procedural language.
We can create and replace db without difficulty.
It is not time consuming process.

SQL uses

1) Data Definition
2) Data retrieval
3) Data Manipulation
4) Access control : Restriction and protection from unauthor
5) Data sharing :- coordinate data sharing by concurrent users.

SQL commands

1) DDL → Data Definition Language
2) DML → Data Manipulation Language
3) DQL → Data Query Language
4) DCL → Data control Language
5) TCL → Transcation control Language.

Page No.
Date

① DDL

used to define Database schema
structure of Database objects in DB.
deals with descriptions of the DB schema.

List of DDL commands :-

~~① create~~

① create → used to create the database or its objects.
② drop → delete objects from the database
③ Alter → change the structure of the db.
④ Truncate → remove all the records from table
⑤ rename → rename an object existing in the db.

② DQL

used to get the data out of the db to perform ops within it.

List of DQL

① Select → It is used to retrieva data from the database.

③ DML

controls the access to data & to the db.
deals with the manipulation of data present in db.

List of DML commands →

① Insert → insert data into table
② update → update existing data in a table.
③ delete → used to delect records from a db table.
④ Lock → table control concurrency
⑤ call → call PL/SQL
⑥ Explain Plan → It describes the access path to data

④ DCL

deals with the rights, permissions, and other controls of the database system.

List of DCL commands :-

1) Grant → gives users access privileges to the db.

Grant select, update on my-table to u1, u2;

2) Revoke → withdraws the user's access privileges given by using Grant command.

Revoke select, update on my-table from u1, u2;

⑤ TCL

- group a set of tasks into a single execution unit.
- Each transaction begins with a specific task and ends when all the tasks in the group successfully complete.
- If any of the tasks fails, transcation fails.
- two results :- success or failure.

List of TCL commands

1) Begin → opens a transaction
2) commit → commits a transcation

commit;

3) rollback → rollbacks a transaction in case of any errors occurs.

rollback;

4) savepoint → sets a save point within a transaction.

savepoint savepoint_name;

Transactions

Transactions, group of a set of tasks int a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks failed, transaction fails. So, thus transaction has only two results :- ① success ② failur

ex. Transcation of $150 to A from B

1. read (B)
2. B := B-150
3. write (B)
4. read (A)
5. A := A+150
6. write(A)

Incomplete steps result in the failure of transaction.

A database transaction, must be Atomic, consistent, isolated, and Durable.

. These properties can ensure the concurrent execution of multiple transactions without conflict.

Implementation of Transaction using SQL.

① BEGIN Transaction :- It indicates the start point of an explicit or local transcation.

Syntax :- BEGIN TRANSACTION Tran_Name ;

② SET TRANSACTION :- Places a name on a transaction

Syntax :- SET TRANSACTION [RO | RW] ;

③ COMMIT :- If everything is in order with all statements in a single transaction, all changes are recorded together in that db is called COMMITTED. The COMMIT command saves all trans. to the DB since the last COMMIT or ROLLBACK cmd.

Syntax :- COMMIT ;

④ ROLLBACK :- If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called ROLLBACK. This command can only be used to undo trans. since last commit or ROLLBACT cmd was issued.

Syntax :- ROLLBACK ;

⑤ SAVEPOINT :- It creates points within the group of transactions in which to rollback. A SAVEPOINT is a point in trans. in which you can roll the transaction back to a certain Point without rolling back the entire trans

Syntax :- SAVEPOINT save_Name ;

⑥ RELEASE SAVEPOINT :- This cmd is used to remove a SAVEPOINT that you have created.

Syntax :- RELEASE SAVEPOINT save_name;

Normalization divides the larger table into smaller & links them using relationships like FK.

Page No.
Date

What is Normalization

Database Normalization is the process of organizing data to minimize data redundancy (data duplication), which in turn ensures data consistency.

Problems of Data Redundancy

1. Disk space wastage
2. Data Inconsistency
3. DML queries can become slow.

Database Normalization is a step by step process. There are 6 normal forms, first Normal form (1NF) through Sixth Normal form (6NF). Most databases are in third Normal Form (3NF) There are certain rules, that each normal form should follow.

First Normal Form (1NF)

A table is said to be in 1NF, if

1. The data in each column should be atomic. No multiple values, separated by values.
2. The table does not contain any repeating column groups.
3. Identify each record uniquely using primary key.

Advantages of Normalization

To minimize data redundancy

data consistency within the db

flexible db design

Disadvantages of Normalization

performance degrades when normalizing the rel's to higher normal forms.

very time consuming

Second Normal Form (2NF)

~~1. The table meets all~~

A table is said to be in 2NF, if

1. The table meets all the conditions of 1NF
2. Move redundant data to a separate table
3. Create relationship b/w these tables using foreign keys

Third Normal Form (3NF)

A table is said to be in 3NF, if

1. It meets all the conditions of 1NF & 2NF.
2. Does not contain columns (attributes) that are not fully dependent upon the primary key.

Functional Dependency

The functional dependency is a rel^nship ~~btw~~ that exists b/w two attributes.

It typically exists b/w the primary key and non-key attribute in table.

$X \longrightarrow Y$

Y is functionally dependent on X

Types of FD

1. Trivial FD
2. Non Trivial FD

1. Trivial FD

$A \rightarrow B$ has trivial FD, if B is a subset of A

eg. $A \rightarrow A$, $B \rightarrow B$.

2. Non-Trivial FD

$A \rightarrow B$ has non-trivial FD, if B is not subset of A.

When $A \cap B$ is NULL, then $A \rightarrow B$ is called complete non-trivial.

BCNF (Boyce Codd Normal Form)

BCNF is the advance version of 3NF.

It is stricter than 3NF.

For BCNF, the table should be in 3NF.

for every FD, LHS is super key.

$X \rightarrow Y$ $X \rightarrow SK$.

Indexing in DBMS

- Indexing is used to optimize the performance of a database by minimizing the number of disk access required where query is processed.
- Used to locate and access the data in the db table quickly.
- Index structure can be created using
  Search key Data Reference columns -
- 1st column → search key → primary key stored in sorted order so that accessed quickly
- 2nd column → data reference → a set of pointer

Indexing Methods

1) Ordered Indices
2) Primary Index → Dense Index / Sparse Index
3) Clustering Index
4) Secondary Index

Ordered Indices

The indices which are sorted are known as ordered indices.

Primary Indices

index → PK → Primary Indexing.
Stored in sorted order

Dense Index	Sparse Index
index record → every search key	index record → only for few
needs more space	

ACID properties plays a vital role in maintaining the consistency and availability of data in the database.

Page No.
Date

Clustering Index

ordered data file.
index → non-PK which may not be unique.
group two or more columns to get the unique value & create index out of them.

Secondary Index

Reduce the size of mapping
Each range is further divided into smaller ranges.
In sparse index, the mapping size grows.

ACID Properties in DBMS

DBMS is the mgmt of data that should be remain integrated when any changes are done it.
If the integrity of the data is affected, whole data get disturbed or corrupted.
So, To maintain the integrity of the Data, there are four properties in DBMS, which are ACID properties

Atomicity
consistency
Isolation
Durability.

Atomicity

data remains atomic.
If any operation is performed on the data,
① either it should be executed completely
② or should not be executed at all.

2) consistency
In DBMS, the integrity of data should be maintained.
→ if a change in db is made → it should remain preserved always.

3) Isolation
→ separation
No data should affect the other one & may occur concurrently.

4) Durability
→ permanency of something.
data after successful execution of operation, become permanent.

Transaction Property
4 properties → maintain consistency in db. → before & after
ACID

1) Atomicity → no midway → run complete → not at all.
It involves two ops: 1) Abort: changes made not visible
2) Commit: changes made visible

2) consistency → integrity constraints → db consistent → b&A
transaction is used to transform db from one consistent state → to another

3) Isolation → concurrency control →
If T_1 is being executed using X data, then X can't be accessed by T_2 until T_1 ends.

4) Durability → Recovery subsystem has responsibility indicate performance of db's consistent state.

Transaction Processing

set of related logical operation.
group of tasks.
An action or series of action.
for accessing the contents of database.

Operations of Transaction

Read(x) –
Read operation used to read the value of x from the database.
And stores it in buffer in main memory

Write(x) –
Used to write the value back to the database from the buffer

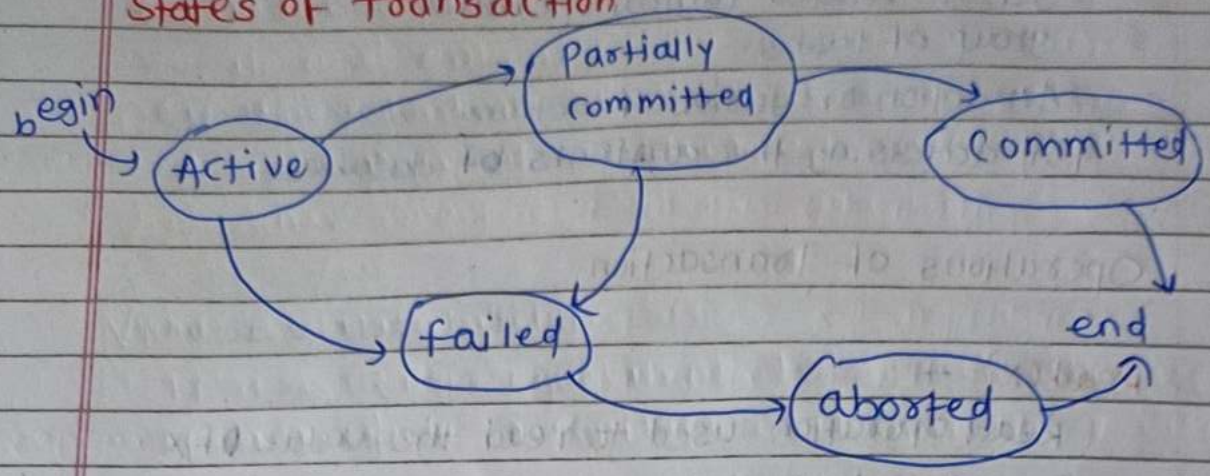
but if may be possible that because of the failure of h/w, s/w or power, etc. that transaction may fail before fished all ops in set.

To solve this problem, we have two important ops:

another **commit :** used to save the work done permanently

Rollback : It is used to undo the work done.

States of transaction



Active State
transaction is being executed

Partially committed
transcation executes in final operation, data is still not saved to db

committed
executes all its ops successfully. data/effects saved permanently in db.

failed state
If any of the checks made by db recovery system fails.

Aborted state
If check fail & Transaction reached → failed state, then db recovery system will make sure that db is in its previous consistent rate. If not, then it will abort or roll back in order to bring db to its consistent rate. After aborting transaction, db recovery module will select one of the following two ops:

1) Restart the transaction 2) kill the transaction

Page No.
Date

Schedule

A series of operation from one Transaction to another.
Used to preserve the order of operation in each of individual transaction.

1. Serial Schedule
One T1 is executed completed before starting another T2.

2. Non-Serial Schedule
- If interleaving of operations is allowed.
- It contains many possible orders in which system can execute individual operations of transaction.

3. Serializable Schedule
- To find non-serial schedules that allow Transaction to execute concurrently without interfering one another.
- If identifies which schedules are correct when interleaving is allowed.
- non-serial ∃ serializable → result ∃ result of its transactions executed serially.

4. Conflict Serializable Schedule
- If after swapping of non-conflicting ops, it can transform into serializable schedule

Conficiting ops
two ops become conflicting if
① both → separate
② same data item
③ atleat one write op.

conflict equivalent
if and only if
① They contain same set of transaction
② each pair of conflict ops are ordered in the same way.

View serializability
If it is view equivalent to serial schedule
If a schedule → conflict, it will be → view.

View Equivalent
Two schedules S1 & S2 are view equivalent if
① Initial Read
→ An Initial Read of both schedules must be same.

② Updated Read
→ In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj

③ Final Write
→ A final write of both schedules must be same.

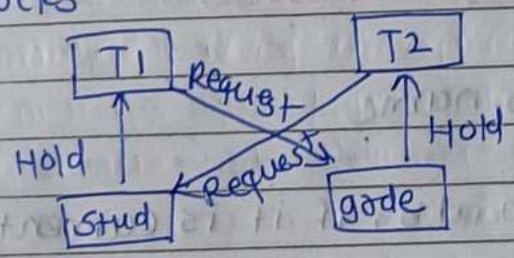
Failure classification

To find that where the problem has occurred, we generalize a failure into

1. Transaction failure
2. System crash
3. Disk failure

Deadlock

A deadlock is a condition where two or more transactions are waiting for one another to give up locks.



Deadlock Avoidance
Deadlock Detection
Deadlock Prevention