Date :
Page No. 2

(-23, +45, +8)
number with sign ↑ | number without sign means +ve (45, 68...) ↑

Signed & Unsigned.

: TINYINT UNSIGNED (0-255) ← wp ↑ range by making tinyint unsigned. the memory which will $ -ve no. to be used are in being used for +ve no.
(max store value { it will take input b/w 0-255)

TINYINT (-128 to 127)
↑ max store value
it will take input b/w (-128-127)

Type of SQL commands

DDL Data Definition Language : create, alter, rename, truncate & drop
DQL Data Query language : select
DML Data manipulation language : insert, update & delete
DCL Data Control language : grant and revoke permission to user
TCL Transaction control language : start transaction, commit, rollback

Primary Key – only 1 Primary key
Unique
Can't be NULL

Foreign Key – multiple foreign key.
duplicate
NULL

(P.K) (F.K)

id	name	city id
101	Karan	1
102	arjun	2
103	ram	1
104	Shyam	3

(1P.K, 2F.K)

(F.K) (P.K)

city	mark
Pune	
Mumbai	
Pune	
Delhi	

id	city-name
1	Pune
2	Mumbai
3	Delhi

(1.P.K)

- Commit ; to save file so that if any misupdate occurs we can reach
- rollback ; to this save stage like backup using rollback

☆ Not NULL, default PK Auto-increment (3)
work on parameterized insert operation
if want Automatic use parameterized
In parameterized insert avoid giving them value so than system Automatically do this insert into st

- id int NOT NULL
- id int UNIQUE
- Gender ENUM('1 MALE', '2 FEMALE', '3 Others') must string. If insert value (1)

way-1 • id int PRIMARY KEY
way-2 • id int,
PRIMARY KEY (id)
way-3 • id int,
name varchar(50),
PRIMARY KEY (id, name)
way-4 • Int Primary Key Auto_Increment
Key must
1, 2, 3, 4 ....
If we insert int 5003 then 5004, 5005, 5006 ....

Create table Student(
id int Primary key Auto-increment
) auto increment = 1200;
1200, 1201, 1202, 1203 ....

used when our key contain duplicate but when we combine 2 attribute we get unique

id	name
101	shayam
101	Ram
102	Ram

Here 101 Shayam
101 Ram
102 Ram
all 3 are different

- Course enum('cds', 'sp') default cds;
- Salary int DEFAULT 25000

Not possible for child change reflect parent
only Parent change reflect child

- cust_id int,
FOREIGN KEY (cust_id) references Customers (id)
(new table (child table)) (old table (Parent table))
old table name + column
on delete, update in parent table will be reflected in child table.
ON DELETE CASCADE
ON UPDATE CASCADE
if must to update, delete in Parent

way-1 • city VARCHAR(50),
age INT,
CONSTRAINT age_check CHECK (age >= 18)
we add constraint to give error a name like here we give age-check so error will be constraint 'age-check is violated without this 'student-chk-1' is violated.
optional
if we try to add age < 18 then change is violated error came.
way-2 • age INT CHECK (age >= 18)
constraint name

way-0 • SELECT age FROM student (display only age column)
way-1 • SELECT age, city FROM student (display only age, city column) (complete)
way-2 • SELECT * FROM student (display all complete existing column)
way-3 • SELECT distinct (only unique row print) city FROM student (display only distinct city by finding unique city in city column)
ALL (It will print duplicate data too)
way-4 • SELECT * FROM student WHERE city = "Mumbai"

Way-5 SELECT name as n from student (it is optional)
Way-5.1 SELECT name n from student
Way-6 SELECT empname from empl;

Way-7 SELECT 44+11;||55

name
abhi
Kuni
Sini

n
abhi
Kuni
Sini

Way-8 SELECT now();
2024-08-20 21:20:25
Way-9 SELECT lcase("abhi");
Way-10 SELECT ucase("abhi");

describe → desc student;

Field	Type	null	Key	Default	Extra
id	int	yes	UNI	NULL	
name	varchar(20)	yes		NULL	
age	int	No		NULL	

(4)

It create same table schema only not data
same info → student
create table if not exists student like info

- Where clause

Like 'r%' // letter start with r
Like '_n%' // letter 2nd start with n

Using operator in where

Arithmetic + - * / %
Comparison =, <> / !=, >, <, >=, <=
(<>, != both are not equal to)
logical AND, OR, NOT, IN (Not IN), BETWEEN (with AND), ALL, LIKE (Not LIKE), ANY
Bitwise & |

Ex- SELECT * FROM student WHERE marks BETWEEN 80 AND 90;
(80 = value1, 90 = value2)

SELECT * FROM student WHERE city IN ("Delhi", "Mumbai");
(matches any value in the list)
WHERE Age>30 AND Dep='IT';
WHERE Age>40 OR Sal>7000;
WHERE NOT Dep='IT';
where grade is NULL;

- Limit clause

n rows → SELECT * FROM Student WHERE marks>80 LIMIT 3
Only first 3 of table will be selected

nth row → SELECT * From student order by id desc limit 2,1;
It will return 3rd row from last.

1	Rahul	8.1
2	Ram	8.2
3	Ravi	7.6
4	Pal	3.4
5	Abhi	9.6
6	mani	9.2
7	Adi	8.5

→ 7 Adi 8.5, 6 mani 9.2, 5 Abhi 9.6

- order by clause
ASC - Ascending (by default if we don't write anything).
DESC - Descending.

Ex- SELECT * FROM student ORDER BY city ASC;

SELECT * FROM student ORDER BY city // It will also order in ascending by default.

On enum order by index
enum('HR', 'Account')
→ Ascending
← Descending

Date :
Page No.

Aggregate Function :- take multiple value and return single value
(perform calculation)

COUNT()
MAX()
MIN()
SUM()
AVG()

Ex- SELECT COUNT(name) FROM STUDENT;

Card	total spend
credit card	₹345
debit card	₹547
upi	₹387

↳ we group by those column which repeats like

GROUP by clause (generally we use group by with some aggregate function)

Ex- SELECT city, avg(marks) FROM student GROUP By city;

SELECT city, name, avg(marks) FROM student GROUP By city, name;

GENERAL ORDER

- SELECT column(s)
FROM table_name
WHERE condition
GROUP BY column(s)
Having condition.
ORDER BY column(s) ASC
Limit

Having Clause (same as where but it is made for group by like split have where clause)

Where :- condition on row
Having - condition after group.

Ex- SELECT count(name), city
FROM student
GROUP By city
HAVING max(marks) > 90;

Date :
Page No.

- set autocommit = 0 // off
  1 // on
- SET SQL_SAFE_UPDATES = 0; // off safe mode so that we can update
- SET SQL_SAFE_UPDATES = 1; // ON safe mode to prevent update

way-1
- UPDATE Student
  SET grade = "O"
  WHERE grade = "A";

way-2 UPDATE Student
SET grade = "B"
WHERE marks BETWEEN 80 AND 90;

way-3
- UPDATE student
  SET marks = marks + 10;

DELETE FROM Student // It will delete those complete row
WHERE mark < 33; Where mark < 33

DELETE FROM student; // It will delete whole table.

TABLE RELATED QUERIES

add constraint
1. ALTER TABLE Student ADD constraint chk_age CHECK (age >= 18); (ADD primary key (column))

add column
2. ALTER TABLE student ADD COLUMN age INT
   optional (New column) (column type)

delete column
3. ALTER TABLE student DROP optional COLUMN age
   DROP Primary Key → two way if Primary key (column) not in 1st int primary key.

rename table name
4. ALTER TABLE Student RENAME TO class

change range of column.
5. ALTER TABLE student MODIFY age VARCHAR(2)

Change column name.
6. ALTER TABLE student CHANGE age stu_age INT;
   name rename type

- ALTER TABLE student add number bigint, drop age;
- ALTER TABLE student add column age int, add column number bigint;
  optional optional

(Put in log and then delete) Delete – can be rollback, delete each row line wise ∴ slower, only specific row delete possible (group of rows)

(direct delete) truncate – can't be rollback, delete all row at once ∴ faster / (7) only specific row delete not possible

Page No.

Truncate → delete table data

DROP → delete table

TRUNCATE TABLE Student;

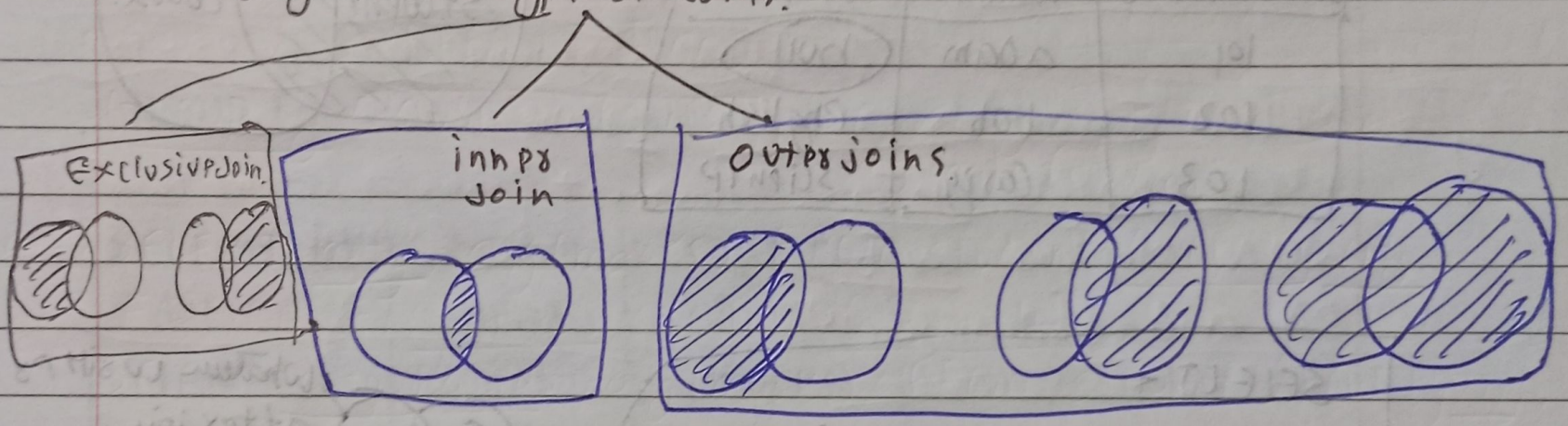
S NO	Name
1	Abhi
2	Shek
3	Kumar

→

S.NO	Name

Joins in SQL :- used to combine rows from 2 or more tables, based on a related column b/w them.

Venn diagram of Type of Joins.



1) Inner Join. (common data b/w two table based on common column value)
(Intersect)

Student

Student_id	name
101	adam
102	bob
103	Casey

course.

Student_id	course
102	English
105	math
103	Science
107	CoS

Way-1

```
SELECT * FROM student as
INNER JOIN course as
ON s.id = c.id;
```

Way-3

```
SELECT * from student
Inner join course
using (student_id);
```

↑ It is used when both table contain same column name

Result

student_id	name	student_id	course
102	bob	102	English
103	Casey	103	Science

Way-2

```
SELECT *
FROM student
INNER JOIN course
ON student.student_id =
course.student_id;
```

Setoppration:- Intersect, Minus, Union (3)
Inner join
Exclusive left
Exclusive right.

Date:
Page No.

2) Left Join (left table data + common data, missing data null) of right side which lies in left table only like id 101

Student

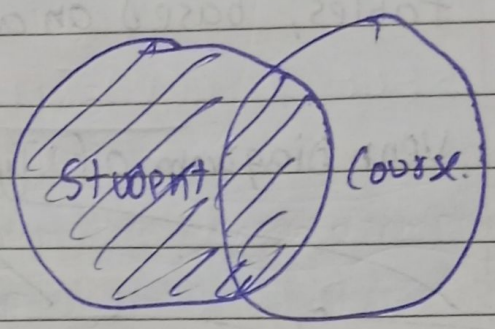
student_id	name
101	adam
102	bob
103	casey

Coursp

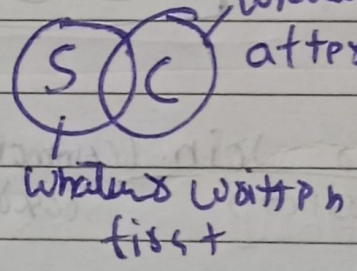
Student_id	coursp
102	English
105	math
103	Sciphip
107	C.S

Result.

Student_id	name	coursp
101	adam	null
102	bob	English
103	casey	Sciphip




```
SELECT *
FROM student as s
LEFT JOIN coursp as c
ON s.student_id = c.student_id;
```

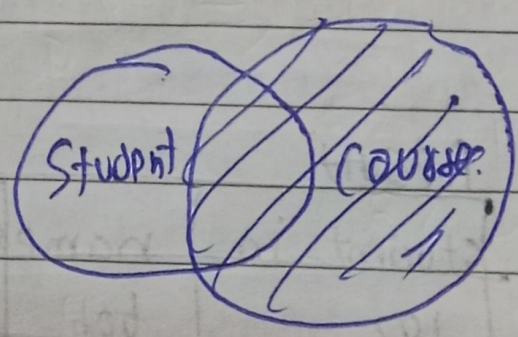


3) Right Join. (right table data + common data, missing data null like above)

RESULT

Student_id	coursp	name
102	English	bob
105	math	null
103	Sciphip	casey
107	C.S	null

```
SELECT *
FROM student as s
RIGHT JOIN coursp as c
ON s.student_id = c.student_id;
```



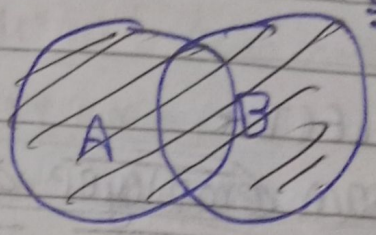
Date :
Page No.

4) FULL Join (doesn't exist in some compiler so we use UNION)

```
SELECT * FROM student as a
LEFT JOIN course as b
ON a.id = b.id

UNION

SELECT * FROM student as a
RIGHT JOIN course as b
ON a.id = b.id;
```



LEFT JOIN
UNION
RIGHT JOIN

RESULT

student_id	name	course
101	adam	null
102	bob	english
103	casey	science
105	null	math
107	null	cs

5) LEFT Exclusive Join
(only left table data, avoid common data)

```
SELECT *
FROM student as a
LEFT JOIN course as b
ON a.id = b.id
WHERE b.id IS NULL;
```

RESULT

id	name	id	course
101	adam	NULL	NULL

6) RIGHT EXCLUSIVE JOIN

```
SELECT *
FROM student as a
RIGHT JOIN course as b
ON a.id = b.id
WHERE a.id IS NULL;
```

RESULT.

id	name	id	course
NULL	NULL	105	math
NULL	NULL	107	c.s

Date :
Page No.

7) Self Join.

Alias must

Way-1
```
SELECT *
FROM employee as a
JOIN employee as b
ON a.id = b.manager_id;
```

id	name	manage_id
103	Casey	NULL
103	Casey	NULL
104	Donald	103

id	name	manager_id
101	adam	103
104	donald	103
102	bob	104

we changed column named as manager from pointing to

Way-2
```
SELECT a.name as manager, b.name
FROM employee as a
JOIN employee as b
ON a.id = b.manager_id;
```

alias must (same as column name given above)

Output

manager	name
casey	adam
donald	bob
casey	donald

Input

id	name	manager_id
101	adam	103
102	bob	104
103	casey	null
104	donald	103

Way-3
```
select * from employee as a,
employee as b where a.id =
b.manager_id;
```

Using join w/o join keyword possible

```
SELECT * from lefttable, Righttable
    where lefttable.id = righttable.id
```

8) Cross Join.

A 1 C B ⇒

A 1	C B
A 1	D 4
B 2	C B
B 2	D 4

SELECT * from students cross join cities;

Date :
Page No.

UNION.

- It is used to combine the result-set of two or more SELECT statements. Gives UNIQUE records.

To use it
- every SELECT should have same no. of columns.
- columns must have similar data types.
- columns in every SELECT should be in same order.

SYNTAX

way-1 SELECT column(s) FROM table A
UNION (unique)
SELECT column(s) FROM table B

way-2 SELECT column(s) FROM table A
UNION ALL (duplicate too)
SELECT column(s) FROM table B.

Input

id (int)	value (varchar(1))
1	A
2	B
3	C

id (int)	value (varchar(1))
1	A
2	B
3	C
4	D

- column order data type are same
- no. of column is same

output - UNION

1	A
2	B
3	C
4	D

UNION ALL

1	A
2	B
3	C
1	A
2	B
3	C
4	D

Date :
Page No.

Subqueries

1) SELECT name, marks
FROM student
WHERE marks > (SELECT AVG(marks) FROM student);
(it will return int value.)

2) SELECT MAX(marks)
FROM (SELECT * FROM student WHERE city = "Mumbai") AS temp;
(it is necessary as we are using subquery in FROM)

3) SELECT (SELECT MAX(marks) FROM student), names
FROM student;
(it is one column i.e it is working.)

RESULT

SELECT MAX(marks) FROM student	name
96	anil
96	chetan
96	dhruv

A use replace → Must need primary key so that it can not duplicate these data. On the basis of primary key it update row rather than inserting

REPLACE :- Insert if data not present else replace data.

- REPLACE INTO customer (id, city) values (125, 'colony');
- REPLACE INTO customer SET id = 1300, [redacted] city = 'utak';
- REPLACE INTO customer (Name, city, id)
  SELECT 'Id, Name, city
  from customer2 where id = 500

(customer2 row updated or inserted in customer table.)

Customer
Abhi Delhi ①

Customer 2
Abhi Noida ①

⇒ Customer
Abhi Noida ①

Date:
Page No.

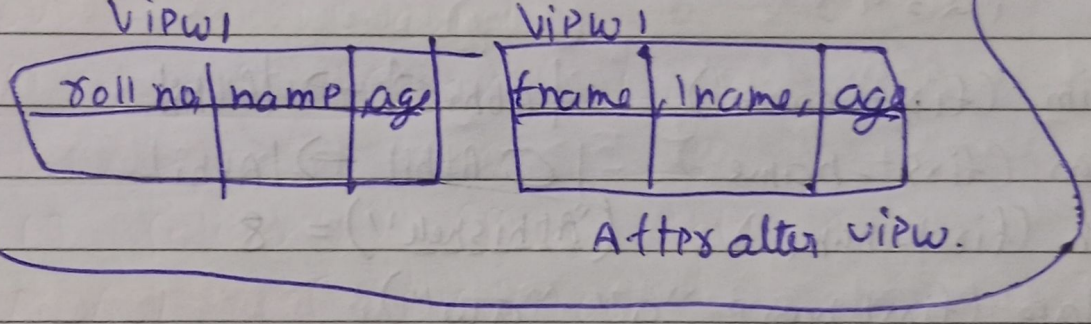
View :- virtual table based on the result-set of an SQL statement.

- CREATE VIEW view1 AS
  SELECT rollno, name, age FROM student;

- SELECT * FROM view1; // how it will show only rollno and name only.

- DROP view If Exists custom view;

- Alter view ~~view~~ view1 AS SELECT fname, lname, age from employee.



Wildcard characters %, _

%Pa%	%abc	Pa%db	%Pabc	%adPa
_Pa_	aPab	bPac		
one one	one one	one one		

SELECT * from worker where first name like '%i%';

Date :
Page No.

user	Host
root	localhost
abhishek	%
abhi	%.umi

DCL

Select user, Host from mysql.user
fixed keyword.

create user

create user abhishek identified by 'abhi@123';
Password
create user abhishek;
optional password
create user abhi@yumi identified by 'mani';
create user 'abhi mani' identified by 'yumi';
optional password.
optional

Drop user
drop user 'abhi mani'
drop user abhi@yumi

Grant

- use college;
  grant select on student
  to abhishek
  privilege
  privilege
- grant select on college.student
  to abhishek
- show grants for abhishek
  Privileges.
- show privileges
- flush privileges. (if any change occurs in maindatabase so its change may not be seen in new user database to reflect change we use it)

Revoke;

- use college;
  revoke select on student
  from abhishek
  privilege
- revoke select on college
  student from abhishek
  privilege.

privileges

Select, Insert, update, delete, drop, alter
All / All privileges.
database / table → create, create view, create user
Show view, show databases
grant option

Date :
Page No.

value condition

case statement (It can be used in select as column, insert, update)

value

update product
set discount = Price
when 100 Then 'Expensive'
when 200 Then 'More Expensive'
else 'unKnown'
END;

→ case statement.

100, 200 single value.

condition

update product
set discount = case
when price > 100 then price * 0.1
when id = 3 then cheap.
else 0
end;

case statement

Price > 100 condition used

id = 2 second condition

* we can use different column and put condition like we choose price, id

- Insert into student value (3, 'Abhi', case statement)
  value condition.

- select name, age, case statement as new column from student
  value condition.

Date :
Page No.

STORE PROCEDURE (Precompiled collection of SQL statement stored in database)

```
delimiter //
create procedure P()
begin
select * from projects;
end//
delimiter ;
call P();
```

TRIGGER (automatically execute specified action in response to certain events (insert/update/delete) on table)

```
delimiter //
create trigger P before insert on x table for each row
begin
set new.c='ht';
end//
delimiter;
```

before/After ← insert - newvalue
before/After ← update - newvalue / old value
delete - old value

```
Savepoint A
rollback to A
```
} After rollback save point delete.

CASE Statement Syntax.

```
update student
   set  name = caseid
        when 1 THEN 'Alice'
        when 2 THEN 'Bob'
        ELSE 'Unknown'
END;
```

( ) from ... → order of running code in sql (It is not like compiler who solve left to right)

Date: / Page No.

WHERE

single row — Select * from emp where deptno = (Select deptno from where dname = 'sales'); (30)

multirow — Select * from emp where deptno IN (Select deptno from dept where dname = 'sales' or dname = 'Research'); (30, 40)

Select max(sal) from emp where sal < (Select max(sal) from emp));

without group by — Select department, count(department) from worker where depar...

multirow with order by clause written in outer query — Select * from emp where deptno IN (Select deptno from dept where dname = 'sales' or dname = 'Research') by deptno desc

Group by

group by — Select department, count(department) from worker group by depar...

We can use outer query table in inner but not viceversa

Select ename, sal from emp e where sal > (Select sal from where e.mgr = m.empo);

Select ename, sal, m.sal from emp e where sal > (Se... from emp m where e.mgr = m.empo); ✗

COLUMN

Select ename, (Select ename from emp m where e.mgr = m... from emp e;

FROM

Select distinct sal from (Select sal, Dense_Rank() (order by sal desc) R from emp) where R=15;

( ) first this is solved then order by clause executed.

(Select * from worker order by worker_id desc limit by worker_id;

Salesman_id: 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007

customer_id: 1, 2, 3, 4, 5, 6, 7, 8

order_no

Sold on -

Sold by -

Date : / /
Page No.

Select firstname, salary from worker where salary = max(salary)

select firstname, salary from worker where salary = (select max(salary) from worker);

first of all inner query solved, 5000 after that value compare with each row in outer query

we need select keyword to use max(salary) function 😊

max(salary) → aggregate

Select dept from worker group by dept;

only dept attribute allowed as group by dept

Select name, dept from worker group by dept;

no other attribute other than dept as group by dept

Select dept, max(salary), min(salary) from worker group by dept

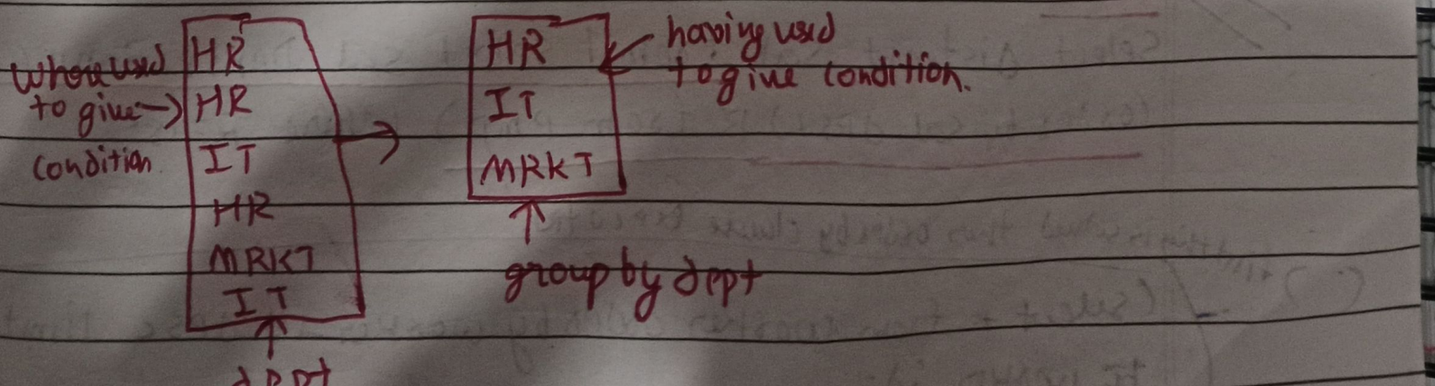
with group by clause we can use aggregate function

Select dept, count(dept) from workers group by department having count(dept) < 4;

with group by clause having is used to give condition.

Select dept, count(dept) from worker group by department where count(dept) < 4;

with group by clause where is not used as where work on each row and we have grouped it where having clause work.



Date :
Page No.

✓ Select first_name, department, salary from worker
Where salary in
(select max(salary) from worker group by dept

50,00,000 , 70,00,000 , 30,000
|| (HR) (MRKT) (IT)

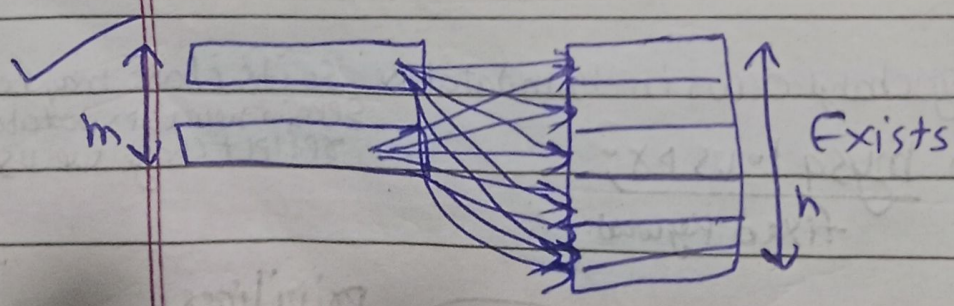
firstname	department	salary
Abhishek	MRKT	70,00,000
Abhi	IT	30,000
Manisha	HR	50,00,000

✗ Select first_name, department, salary from worker
Salary in — we are using in that take single column [3, 5, 7, 8]
(select dept, max(salary) from worker group by dept

It will return.

HR	50,00,000
MRKT	70,00,000
IT	30,000

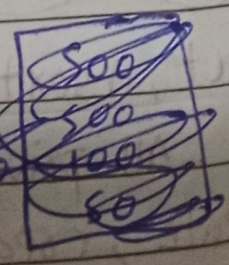
there are two column so in will not work on it



Total comparision = m x n.

Select * from where e
(select id from dept
emp.e_id = dept.

Select * from emp where salary = (select max(salary) from emp)



count (salary) = 5

count (*) = 6

count (distinct salary) = 4

1	10
2	20
3	30
4	40
5	30

Select firstnam

Select firstnam
max(salary) f
we need sele

Select dept
only d
allow

Select name
1
no oth
other
gro

Select dept
by dept

Select dept,
having count
with gr

Select dept
Where count
with grou
each row

who used to give → condition	HR
	HR
	IT
	HR
	M
	I

d

Date : / /
Page No.

Select user, Host from mysql.user

user	Host
root	localhost
abhishek	%
abhi	Yumi

DCL

create user

create user abhishek identified by 'abhi@123';
create user abhishek;
create user abhi@Yumi identified by 'mani';
create user 'abhi mani' identified by 'Yumi';

optional

Drop user

drop user 'abhi mani'
drop user abhi@Yumi

Grant

use college;
grant select on ~~college~~ student. to abhishek
grant select on college.student to abhishek

privileges

Revoke

- use college;
  revoke select on student from abhishek
- revoke select on college.student from abhishek

privileges.

- show grants for abhishek.
- show privileges
- flush privileges (if any change occurs in main database, so its change may not be seen in new user's database to reflect change we use it)
- select user, Host from mysql.user;

fixed keyword.

privileges.

select, insert, update, delete, drop, alter
All / All privileges.
create, create view, create user
show view, show databases.
grant option.

SQL Merge Statement.

```
Merge Target_table_name as t
using source_table_name as s
 on t.productid = s.productid
When matched then update
set t.price = s.price, t.productname = s.productname.
When not matched by target then
insert values (s.productid, s.productname, s.price)
When not matched by source then delete;
```

* change will be seen in target_table
- If same id found in source and target and any change found in source then target row of same id will be updated using source
- If id found in source but not in target then source row. will be inserted in target.
- If id not present in source but exist in target then delete that row in target.

Source_table

ProductId	ProductName	Price
1	Table	90
3	Chair	70

Target_table

ProductId	ProductName	Price
1	Table	~~100~~ 90 (update) ①
~~2~~	~~Desk~~	~~180~~ (delete) ③
3	chair	70 (insert) ②

operation ① ② ③

11 Output

Source_table

ProductId	ProductName	Price
1	Table	90
3	Chair	70

Target_table

ProductId	ProductName	Price
1	Table	90
3	chair	70