Day_6

create a emp table

		EMP			
EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

create a dept table

	DEPT	
DEPTNO	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

```
MySQL - SQL Joins

select deptno, sum(sal) from emp
group by deptno;

DEPTNO   SUM(SAL)
------   --------
1        18000
2        17000
```

```
select dname, sum(sal) from emp, dept
where dept.deptno = emp.deptno
group by dname;

DNAME    SUM(SAL)
-----    --------
TRN      18000
EXP      17000

select upper(dname), sum(sal) from emp, dept
where dept.deptno = emp.deptno
group by upper(dname)
having sum(sal) > 10000
order by 1;
```

Types of Joins:

select dname, ename from emp, dept

where dept.deptno = emp.deptno;

1. Equi Join(Natural join)

- join based on equality condition
- it shows matching rows of both the tables
- uses:

  a. DNAME, ENAME,

  b. SNAME, CNAME

- most frequently used join (>90%), hence it’s also know as Natural join.

```
dept -> driving table
emp -> driven table

                                    <<---
select dname, ename from emp, dept
where dept.deptno = emp.deptno;
```

DNAME	ENAME
TRN	Arun
TRN	Ali
TRN	Kirun
EXP	Jack
EXP	Thomas

select dname, ename from emp, dept

where dept.deptno != emp.deptno;

2. Inequi Join(Non-equijoin)

- join based on inequality condition
- it shows me no matching rows of both the tables
- use: a. Exception Reports

    b. Who are the employees who don’t belong to TRN

    c. who are customers who haven’t made the payment

```
select dname, ename from emp, dept
where dept.deptno != emp.deptno and dname = 'TRN';

DNAME    ENAME
-----    -----
TRN      Jack
TRN      Thomas
```

3. Outer Join

- join with (+) sign or if the keyword outer
- shows matching rows of the tables + non matching of “Outer” table
- Outer table → table which is on Outer side of (+) sign.
- Outer table → table which is Opposite side of (+) sign.
- use: a)Master-Detail Report(Parent-Child Report)
- Half Outer Join: (1 do-while, 1 for loop)

  (+) sign on any one side

  1. Right Outer Join
  2. Left Outer join
- Full Outer Join:(nested do-while loop)
  - Shows matching rows of both the tables + non-matching rows of both the tables
  - UNION of Right Outerjoin and Left Outerjoin

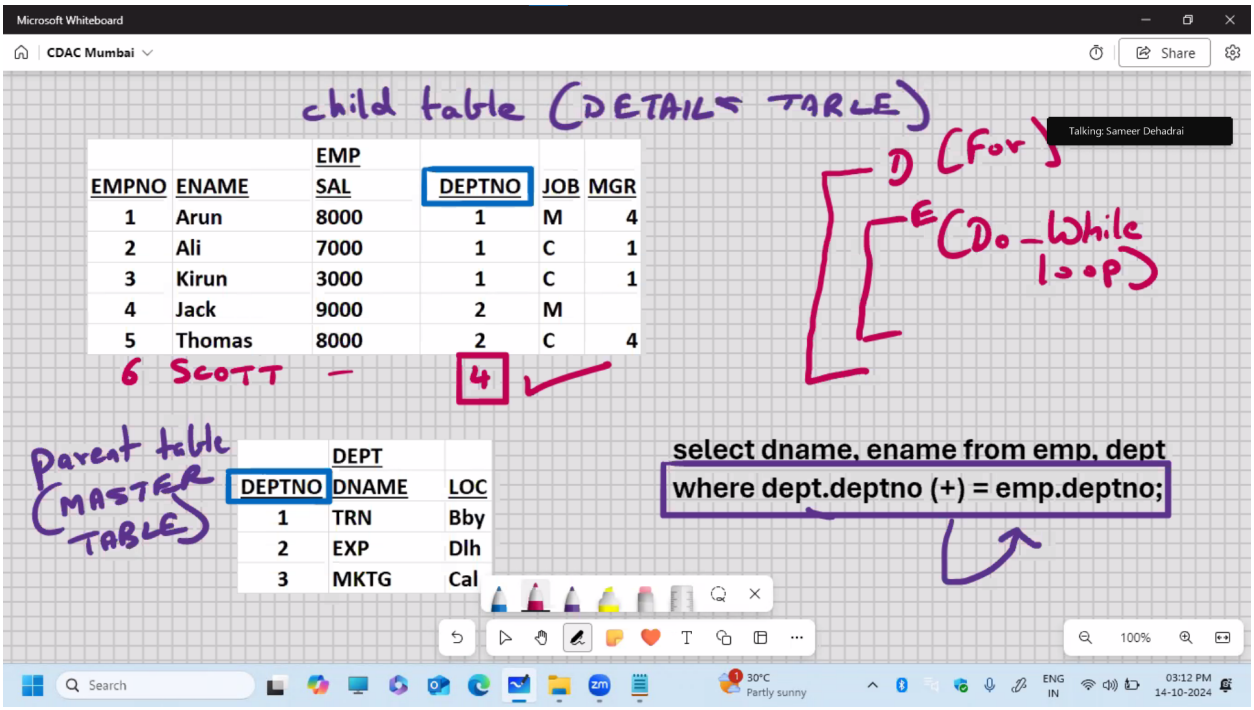
  (+) sign on both the side

```
select dname, ename from emp, dept
where dept.deptno = emp.deptno (+);
```

Right Outer Join

```
select dname, ename from emp, dept
where dept.deptno (+) = emp.deptno;
```

Left Outer Join




```
select dname, ename from emp, dept
where dept.deptno = emp.deptno (+)
union
```

```
ANSI syntax for Full Outerjoin:-

select dname, ename from emp full outer join dept
on (dept.deptno = emp.deptno);
-----------------------------------------------------

ANSI syntax for Right Outerjoin:-

select dname, ename from emp right outer join dept
on (dept.deptno = emp.deptno);
-----------------------------------------------------

ANSI syntax for Left Outerjoin:-

select dname, ename from emp left outer join dept
on (dept.deptno = emp.deptno);
```

- (+) is not supported by MySQL
- ANSI syntax for right outer join is supported by MySQL
- ANSI syntax for left outer join is supported by MySQL
- ANSI syntax for Full outer join is not supported by MySQL

```
To  implementFull Outerjoin in MySQL:-

ANSI syntax for Right Outerjoin:-
*       UNION of ANSI syntax for Right Outerjoin and ANSI
        syntax for Left Outerjoin

select dname, ename from emp right outer join dept
on (dept.deptno = emp.deptno)
        union
select dname, ename from emp left outer join dept
on (dept.deptno = emp.deptno);
```

- INNER JOIN → by default every join is Inner join; putting a (+) sign or using the keyword ‘Outer’ is what makes it an Outer Join

4. Cartesian join(Cross join)
    - select dname, ename from emp, deptno;
    - Join without a WHERE clause
    - every row of driving table is combined with each and every row of driven table
    - it takes the Cross product of 2 tables and therefor its also know as Cross Join.

```
select dname, ename from emp, dept;          <- FAST (LESSER THE I/O
                                             BETWEEN DB SERVER HD AND
                                             SERVER RAM, THE FASTER IT
                                             WILL BE)

select dname, ename from dept, emp;          <- SLOW (MORE THE I/O
                                             BETWEEN DB SERVER HD AND
                                             SERVER RAM, THE SLOWER IT
                                             WILL BE)
```

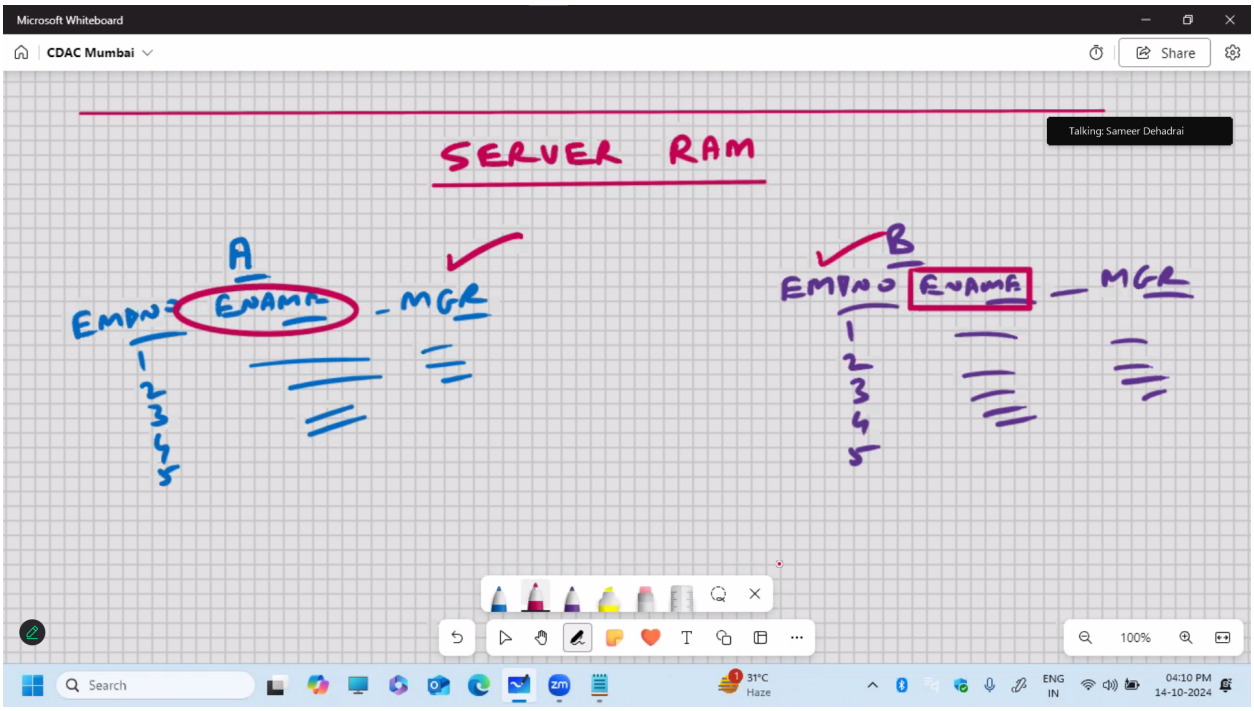
    - use: a) for printing purposes e.g In students table you all the students names, in subjects table you have all the subjects names; when you are pritning the mark-sheets for the students, every student name is combined with each and every subject name.

5. Self Join

- joining a table to itself
- used when parent column and child column, both are present in the same table
- slowest join(based on Recursion)
- uses: a) Ename, MGR name

select a.ename, b.ename from emp b, emp a where a.mgr = b.empno;

```
select a.ename, b.ename from emp b, emp a
where a.mgr = b.empno;
```



- Cartesian join is the Fastest Join, because there is no where clause, and therefore there is no searching involved

- JOINING 3 or MORE TABLES

create a tables

	DEPTHEAD
DEPTNO	DHEAD
1	Arun
2	Jack

	PROJECTS	
PROJNO	CLIENT_NAME	DESCRIP
P1	Deloitte	CGS
P2	Morgan Stanley	AMS
P3	BNP Paribas	Macros
P4	ICICI Bank	PPS
P5	AMFI	Website Devl

```
                                    (5)   (3)    (2)
select dname, ename, dhead from emp, dept, depthead
where depthead.deptno = dept.deptno
and dept.deptno = emp.deptno;
```

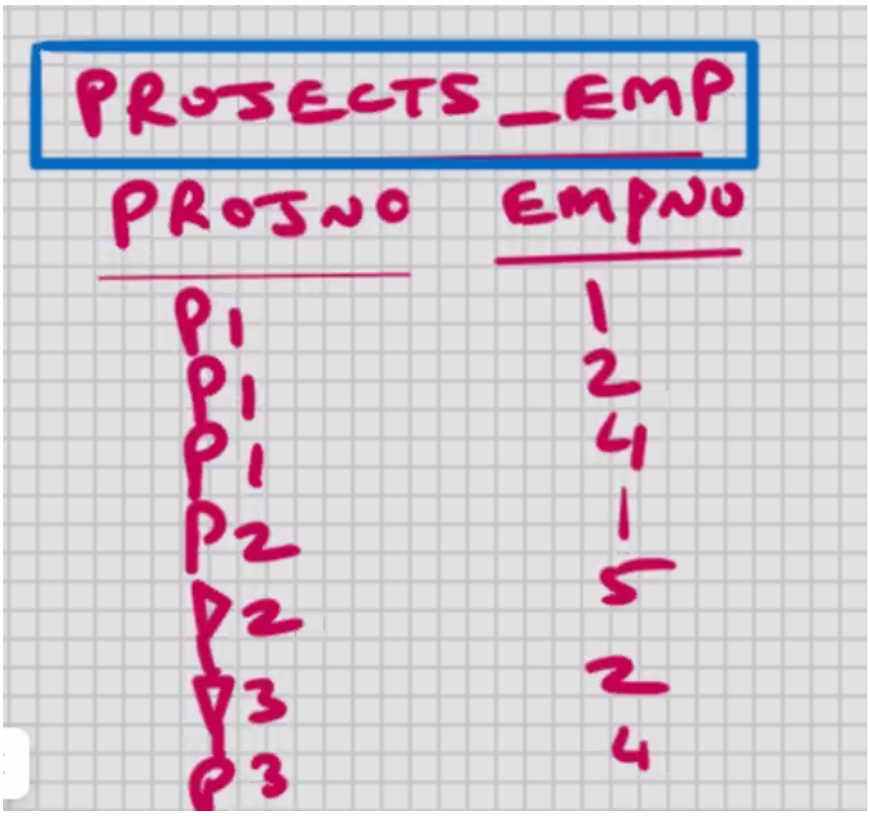
Types of Relationships:

1 : 1 (dept : depthead)or(depthead : dept)

1 : Many (Dept : Emp) and (DeptHead : Emp)

Many : 1 (Emp : Dept) and (Emp : DeptHead)

Many : Many (Projects : Emp) or (Emp : Projects)



Intersection Table

- Intersection table is required for Many : Many Relationship:

```
select client_name, ename from projects_emp, emp, projects
where projects_emp.projno = projects.projno
and projects_emp.empno = projects.empno
order by 1, 2;
```

MySQL - SQL - Sub-queries:

- Nested queries (Query within Query) (Select within select)

		EMP			
EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

- minimum sal:

select min(sal) from emp;

select ename, min(sal) from emp; ← error

select ename from emp where sal = min(sal); ← error

- select ename from emp where sal=(select min(sal) from emp);

```
Display the ENAME who is receiving SAL = min(sal):-

                                                    Talking: Sameer Dehadrai

select ename from emp                    <- main query (parent) (outer)
where sal =
(select min(sal) from emp);              <- sub-query (child) (inner)
```

- max upto 255 levels for sub-queries(this limit of SQL can be exceeded with the help of Views)
- Join is faster than sub-query because when you write a join you solve a problem using one select statement, whereas with sub-queries you require 2 or more select statements
- the more the number of select statements the slower it will be

Display the 2nd largest SAL:-

```
select max(sal) from emp
where sal <
(select max(sal) from emp);
```

Display all the rows who belong to the same DEPTNO as 'Thomas':-

```
select * from emp
where deptno =
(select deptno from emp
where ename = 'Thomas');
```

```
Using sub-query with DML commands:-
In Other RDBMS:-

delete from emp
where deptno =
(select deptno from emp
where ename = 'Thomas');


update emp set sal = 10000
where job =
(select job from emp
where ename = 'Kirun');
```

- The above 2 commands will not work in MySQL
- In MySQL you cannot update or delete from a table from which you are currently selecting

```
Using sub-query with DML commands:-
Solution for MySQL:-

delete from emp
where deptno =
(select abcd.deptno from
(select deptno from emp
where ename = 'Thomas') abcd);

update emp set sal = 10000
where job =
(select pqr.job from
(select job from emp
where ename = 'Kirun') pqr);
```

Multi-row Sub-Queries (sub-query return multiple rows):

```
Multi-row sub-queries (sub-query returns multiple rows):-

Display all the rows who are receiving a SAL = any of the Managers:-


select * from emp
where sal =any          (8000,9000)
(select sal from emp
where job = 'M');

select * from emp
where sal in            (8000,9000)
(select sal from emp
where job = 'M');
```

```
select * from emp
where sal >=                (8000)
(select min(sal) from emp
where job = 'M');
```

To make it work faster:-

1. Try to solve the problem using a Join, because Join is faster than sub-query
2. Try to reduce the number of levels for sub queries
3. Try to reduce the number of rows returned by sub-query

```
Assumption, 3rd row SAL is 13000:-


select * from emp
where sal > all          (8000,9000,....)
(select sal from emp
where job = 'M');

select * from emp
where sal >              (9000)
(select max(sal) from emp
where job = 'M');

ALL -> logical AND
ANY -> logical OR
IN  -> logical OR
```

```
*       Assumption 3rd row SAL is 3000

Using sub-query in the HAVING clause:-
Display the DNAME that is having max(sum(sal)):-

select deptno, sum(sal) from emp
group by deptno;

deptno  sum(sal)
------  --------
1       18000
2       17000
```

```
select sum(sal) from emp
group by deptno;

sum(sal)
--------
18000
17000
```

```
select sum(sal) sum_sal from emp
group by deptno;

sum_sal
-------
18000
17000
```

```
select max(sum_sal) from
(select sum(sal) sum_sal from emp
group by deptno) abcd;

max(sum_sal)
------------
18000
```

```
select deptno, sum(sal) from emp
group by deptno
having sum(sal) =
(select max(sum_sal) from
(select sum(sal) sum_sal from emp
group by deptno) abcd);

deptno  sum(sal)
------  --------
1       18000
```

```
select dname, sum(sal) from emp, dept
where dept.deptno = emp.deptno
group by dname
having sum(sal) =
(select max(sum_sal) from
(select sum(sal) sum_sal from emp
group by deptno) abcd);

dname   sum(sal)
-----   --------
TRN     18000

*       INLINE VIEW -> if you have a sub-query in the FROM clause, then
        it's known as Inline View
```

Talking: Sameer Dehadrai

Correlated sub-query (using EXISTS operator):

- this is the exception when sub-query is faster than Join

Display the Dname’s that contain employees:

Solution #1:

select deptno from emp;

1

1

1

2

2

select distinct deptno from emp;

1

2

select dname from dept where deptno = any(select distinct deptno from emp);

TRN

EXP

select dname from dept where deptno in(select distinct deptno from emp);

TRN

EXP

select dname from dept where deptno not in (select distinct deptno from emp);

MKTG

Solution #2:

select dname from emp, dept where dept.deptno = emp.deptno;

TRN

TRN

TRN

EXP

EXP

select distinct dname from emp, dept where dept.deptno = emp.deptno;

TRN

EXP

Solution #3:

select dname from dept where exists (select deptno from emp where dept.deptno = emp.deptno);
← faster

TRN

EXP

- first the main query is executed
- for every row returned by the main query, it will run the sub-query once
- sub-query returns bollean TRUE value or FALSE value
- if sub-query return TRUE value, then main query is eventually executed for that row
- if sub-query return FALSE value, then main query is not executed for that row
- inlike earlier, we do not use DISTINCT here, therefore no sorting takes place in Server RAM; this speeds it up

- unlike a Join, the number of full table scans is reduce; this further speeds it up

select dname from dept where not exists (select deptno from emp where dept.deptno = emp.deptno);

MKTG