DBMS Lecture 9

Subqueries:

The main benefit of subquery is: - Easy to read and understand complex queries.

Example1: Let's take students table,

Q- Find all the students who have a psp greater than psp of student with id=18.

```
 select *
from students a
join students b
on a.psp>b.psp
and b.id=18;
```

A join basically works like, There are 2 tables, it pick 1 row from table1 and try to evaluate the codition against every row of the second table. This is correct. But if someone read this query it's not very easy to understand.

Subqueries: - They are rarely used to do things not otherwise doable. - But mostly used to do things in a more understandable or intuitive way. - Most of the things that we can do with subqueries can be done via Join as well.

What a begineer would have done? 1. Get psp of student with id =18(87) 2. Get students with psp> 87.

```
 select psp
from students
where id=18;
```

+

```
 select id
from students
where psp> 87;
```

Combine them together.

```
 select id
from students
where psp> (
select psp
from students
where id=18);
```

Just by reading it is understandable.

Now, how it works: Same as select query works, It will first go to student but here the condition is a complete query in itself. So it goes to every row and then for every row it is going to execute that query.

For every row of student, execute the query.

Side effects: Performance wise slower. T.C=> N^2^

Example2: Let's have students table and TA table,

Q- Print the name of students who are a TA as well.

Joins:

```
 select s.name
from TA t
join students s
on t.st_id=s.id;
```

IN Clause:

When we have to compare something, if a value is in the list of values we have done this: id IN (1,6,7). *Now, using the same thing here:*

```
 select name
from students
where id IN (
select st_id
from TA
);
```

This query will give the list of values.

==NOT IN can also be used.==

ALL:

Q- Select students who have psp greater than every student of batch_id=3.

```
 select *
from students
where psp > (
select max(psp)
from students
group by batch_id
having batch_id=3
);
```

```
 select
from students
where psp > ALL(
select psp
from students
where batch_id=3
)
```

1. It's going to get multiple values
2. Match the condition against all values.
3. Only if all match, output that.

We got 4 psp(87,92) Then match to all of these, only if all of these return true, then output that.

ANY:

```
 select
from students
where psp > ALL(
select psp
from students
where batch_id=3
)
```

- Output the row if any of the subqueries row match.

CORELATED SUBQUERIES:

Q- Select all students who have a psp greater than average psp of their batch.

```
 select avg(psp)
from students
group by batch_id={If we know the batch_id}
```

```
 select name
from students
where psp>{Upper query}
```

```
 select name
from students s
where psp> (
select avg(psp)
from students
group by batch_id
having batch_id=s.batch_id
);
```

Upper query had a variable that can be referred by a subquery.

EXISTS

```
 select *
from students s
where EXISTS (
SELECT *
from TA
where TA.st_id=s.id
)
```

Subquery will return a TRUE if even a single row is present.

In above query, IN CLAUSE one we checked for every student if it is a TA or not here it will return >1 rows and then we will check against that rows. whereas in EXISTS CLAUSE as soon as even a single row is returned it will return true.

==EXISTS is much faster than IN.==

Till now subqueries have been used in WHERE clause. Subqueries can also be used in SELECT also in FROM.

SUBQUERY USING SELECT

If av_psp was not there, then

```
 select id,name,psp
from students
```

But now, for every student we need to find average psp. So it's kind of a query in itself.

```
 select id,name,b_id,(select avg(psp)
from students
group by b_id
having b_id=s.b_id)
from students s;
```

for every student s: 1. execute the query 2. put the cell value along with the row.

SUBQUERY USING FROM:

- Output of every query is kind of a table in itself.

```
 select student.name as name, student.id as id, student.psp as psp
from student;
```

Example:

Output: student_name, instructor_name.

```
 select s.name as s_name, b.id as b_id,i.name as ins_name
from students s
join batches b
on s.b_id=b.id
join instructors i
on b.ins_id=i.ins_id;
```

OUTPUT: table is t, columns: s_name, b_id, ins_name

If this is input given to us, output we want is: s_name, ins_name

```
 select s_name, ins_name
from t;
```

t is nothing but output of the query..,

So, *select from (complete query here)*

```
 select t.s_name, t.ins_name
from (
select s.name as s_name, b.id as b_id,i.name as ins_name
from students s
join batches b
on s.b_id=b.id
join instructors i
on b.ins_id=i.ins_id
) t;
```

USE Case:

1. P1 wants to know name of instructor for every student
2. P2 wants to know name of instructor for every batch
3. P3 wants to know name of batch for every student.

So we sometimes don't want to give access of complete table, let's say we create a table as `not_normalized_for_ba`

Create a mega table for everything so that any business analyst can perform their queries on.

But As a developer: We will have to store redundant data.

Here comes Views:

VIEWS:

- Views are kind of an abstraction over real tables.
- They will provide a visibility to data that we want to provide.
- Act like tables while no data is actually stored in them.

In MYSQL Workbench:

Output:

order_id, order_date, customer_id, first_name, points, product_id, product_name, unit_price, quantity.

```
 select db.order_id, db.first_name from
(select o.order_id, o.order_date, o.customer_id, c.first_name, c.points,
p.product_id, p.name, p.unit_price, oi.quantity
from sql_store.orders o
join sql_store.customers c
using(customer_id)
join sql_store.order_items oi
using(order_id)
join sql_store.products
using(product_id)) as db;
```

So we have shared this table with everyone. So the code will be getting repeated, after few days we change name of columns or something in these tables so it will have to be changed in all the places. So this is not good.

So we can actually create an illusion of a table using this query.

```
 Create view order_product_customers as
select o.order_id, o.order_date, o.customer_id, c.first_name, c.points,
p.product_id, p.name, p.unit_price, oi.quantity
from sql_store.orders o
join sql_store.customers c
using(customer_id)
join sql_store.order_items oi
using(order_id)
join sql_store.products
using(product_id);
```

Now the place where actual table is stored internally is secured, no one has access to that table directly.

If someone need to make a change in the table and just ensure that the names of the column in the table will be same then,

create or replace view as {paste the new query}

So view is like a table but it is not storing any data.

Now, Can we use CRUD opeartion in views?

- Read

```
 select opc.order_id, opc.firstname
from sql_store.order_product_customer opc;
```

- Update:

```
 update sql_store.order_product_customer opc
set opc.first_name = 'naman'
where opc.order_id=1;
```

It's succeded but in customers table there is a customer called naman in middle because the customer who placed the order with id=1 has the customer id=6 so their firstname change to naman.

So views also can allow update on the internal tables, so if we dont want that, Do ensure that whenever we create a view do set the restrictions on the view.

So, Can we also create a row using views? Actually it depends.

- Create is only possible if we want to create in a single table.
- Mostly views are only used to do a read.
- But never for create and delete.

Example1:

Placement policy of scaler: If a student psp>=80 eligible for placement else not.

students table: id, name,psp **output:** id,name,eligible_for_placements(yes/no).

If psp>=80,output= yes else output=no. There is something in programming languages that, given a condition if true return this else this: operator called ternary operator. ▫

In sql:

IF: *if(condition ,valueontrue, valueonfalse)*

```
 select id,name, if(psp>=80, "yes","no") as eligible_for_placements
from students;
```

Example2:

We want priority order of placements.

If someone has: *80%+: PO [60-80): P1 <60: P2*

Output: id,name,placement_priority.

In programming language we use Switch case:

In sql:

Case: CASE WHEN {condition} THEN {output} WHEN {condition} THEN {output} WHEN {condition} THEN {output} ELSE {output} END as {name}

```
 select id,name,
case
when psp>=80 then "P0"
when psp>=60 then "P1"
else "P2"
end as placement_priority
from students;
```

If a column has NULL value, Output something else.

Example: For every student give their psp. (some students may have psp NULL. If NULL put their psp as 0)

```
 select IF NULL(psp,0)
from students;
```

It's saying if the psp is not null, output the value of psp else output this value(0).

- Output psp of every student, if NULL, output number of assignments questions solved, if it is also NULL output number of homework questions solved else output 0.

`ifnull(psp, ifnull(assignments_questions, ifnull(homework_questions,0)))`

COALESCE:

`coalesce(psp,assn,hw,0);` It has priority order of values, first try this then this then this.. ==Stack of NULL checks.==

```
 select id,name, coalesce(psp,assgn,hw,0)
from students;
```
