SQL – Subqueries

Dr. Villanes

Questions
Q1 – Q2

Question 1

How many rows and columns are returned from this query?

```
select *
   from customer2, transaction2;
```

customer2

ID	Name
101	Jones
101	Jones
102	Kent
102	Kent
104	Avery

transaction2

ID	Action	Amount
102	Purchase	$376
102	Return	$119
103	Purchase	$57
105	Purchase	$98

A	4 rows, 5 columns
B	20 rows, 5 columns
C	9 rows, 5 columns

Question 2

Given these two SELECT statements, will the result sets contain the same data?

```
select *
   from customers left join transactions
   on customers.ID=transactions.ID;
```

```
select *
   from transactions right join customers
   on customers.ID=transactions.ID;
```

A	Yes
B	No

ANSI Standard

One solution is to repeat the calculation in the WHERE clause.

```
select Employee_ID, Employee_Gender,
       Salary, Salary*.10 as Bonus
   from jupiter.employee_information
   where Salary*.10<3000;
```

ANSI standard

What does the Postgres manual say?

- SQL standard: column aliases can be referenced in ORDER BY, GROUP BY and HAVING clauses.

- **However**, in Postgres: An output column's name can be used to refer to the column's value in ORDER BY and GROUP BY clauses, but not in the WHERE or HAVING clauses; there you must write out the expression instead.

- Reference: https://www.postgresql.org/docs/current/sql-select.html#SQL-SELECT-LIST

Subqueries

(also known as inner queries, inner select or nested queries)

What is a Subquery?

A subquery:

- is a **query within another SQL query**
- returns values to be used:
    - **SQLite:** You can use a subquery in the SELECT, WHERE, or JOIN clause.
    - **Postgres:** You can use a subquery in the SELECT or WHERE clause
- must return only a single column
- can return multiple values or a single value.

Example of a Subquery

outer query

subquery

```
SELECT trackid, name, albumid
FROM tracks
WHERE albumid = (SELECT albumid
                 FROM albums
                 WHERE title =
         'Let There Be Rock');
```

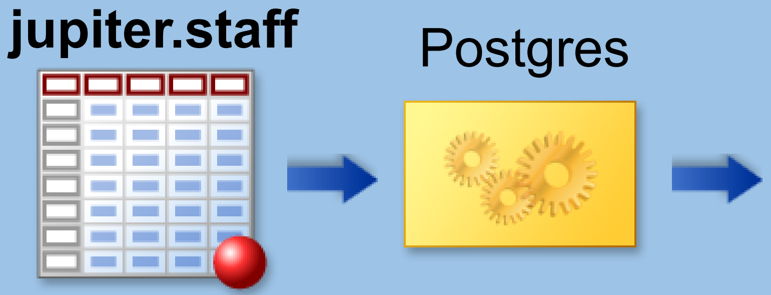
Two Types of Subqueries

There are two types of subqueries:

- A *noncorrelated subquery* is a self-contained query. It executes independently of the outer query.
- A *correlated subquery* requires a value or values to be passed to it by the outer (main) query before it can be successfully resolved.

Non-correlated query example

Generate a report that displays **Job_Title** for job groups with an average salary greater than the average salary of the company as a whole.



```
Employee Job Title           MeanSalary
----------------------------------------
Account Manager                   46090
Administration Manager            47415
Applications Developer I          42760
 ...
```

Step 0: Create a temporary table

1. Calculate a temporary table **"Jupiter".Temp_Staff**

```
CREATE TABLE "Jupiter".Temp_Staff as
select *, CAST(REPLACE(REPLACE("Salary",'$',''),',',''),
AS INTEGER) as Int_Salary from "Jupiter".staff;
```

Step 1: Subquery

1. Calculate the company's average salary

```
select avg("int_salary") as CompanyMeanSalary
   from "Jupiter".Temp_Staff;
```

Step 2: Outer Query

2. Determine the job titles whose average salary exceeds the company’s average salary.

```
select "Job_Title", avg("int_salary") as MeanSalary
from "Jupiter".Temp_Staff
group by "Job_Title"
having avg("int_salary")>38041.51;
```

Step 3: piece it together

The subquery is resolved before the outer query can be resolved

```
select "Job_Title", avg("int_salary") as MeanSalary
from "Jupiter".Temp_Staff
group by "Job_Title"
having avg("int_salary")>(select avg("int_salary") as
     CompanyMeanSalary
     from "Jupiter".Temp_Staff);
```

Evaluate the subquery first.

Correlated query example

A *correlated subquery* requires a value or values to be passed to it by the outer (main) query before it can be successfully resolved.

```
select Employee_ID, avg(Salary) as MeanSalary
   from employee_addresses
   where 'AU'=
      (select Country
          from supervisors
          where employee_addresses.Employee_ID=
                supervisors.Employee_ID)
group by 1;
```

This query is not stand-alone.
It needs additional information from the main query.

Returning multiple rows from the subquery

A subquery can return multiple values or a single value.

However, subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.

```
select Employee_Name, City, Country
from employee_addresses
where Employee_ID IN
      (select Employee_ID
          from employee_payroll
          where Birth_Month=2)
order by 1;
```

The NOT IN operator displays a record if the condition(s) is NOT TRUE.

In-Line Views

(also known as subquery)

What is an In-Line View?

An *in-line view* is a query expression (SELECT statement) that resides in a FROM clause:

- It acts as a virtual table, used in place of a physical table in a query.
- An in-line view can return more than just one column

```
SELECT sub.*
FROM (SELECT date, location, resolution
         FROM tutorial.sf_crime_incidents
         WHERE day_of_week = 'Friday')as
         sub
WHERE sub.resolution = 'NONE'
```

In-Line View Exercise

List all active Sales Department employees who have annual salaries significantly lower (less than 95%) than the average salary for everyone with the same job title.

```
                                              Employee
                                                Annual
Employee_Name         Employee Job Title        Salary   Job_Avg

Ould, Tulsidas        Sales Rep. I              22,710    26,576
Polky, Asishana       Sales Rep. I              25,110    26,576
Voron, Tachaun        Sales Rep. I              25,125    26,576
```

Step 1

Calculate the average salaries for active employees in the Sales Department, grouped by job title.

Tables: **`employee_payroll, employee_organization`**

Conditions: **`"Employee_Term_Date" is null & "Department"='Sales'`**

```
select "Job_Title",
       avg("Salary") as Job_Avg
      from "Jupiter".employee_payroll as p,
        "Jupiter".employee_organization as o
   where p."Employee_ID"=o."Employee_ID"
         and "Employee_Term_Date" is null
         and "Department"='Sales'
   group by 1;
```

Step 2

Match each employee to a job title group and compare the employee's salary to the group's average to determine whether it is less than 95% of the group average.

```
select "Employee_Name", emp."Job_Title",
       "Salary", job.Job_Avg
from (select "Job_Title",
       avg("Salary") as Job_Avg
       from "Jupiter".employee_payroll as p,
       "Jupiter".employee_organization as o
       where p."Employee_ID"=o."Employee_ID"
       and "Employee_Term_Date" is null
       and "Department"='Sales'
       group by 1) as job, "Jupiter".salesstaff as emp
where emp."Job_Title"=job."Job_Title"
       and CAST(REPLACE(REPLACE("Salary",'$',''),',',''))
AS INTEGER)<job.Job_Avg*.95
         and "Emp_Term_Date" is null
order by 1;
```

Lab #3

- Reminder: in Postgres, an output column's name can be used to refer to the column's value in ORDER BY and GROUP BY clauses, but not in the WHERE or HAVING clauses; there you must write out the expression instead.
- Question 2a: you might want to use EXTRACT(MONTH from XXX)
- Question 4: you don’t need to use a subquery for this one. If you need to join people and people_movies, please refer back to the last slide on Class #3.