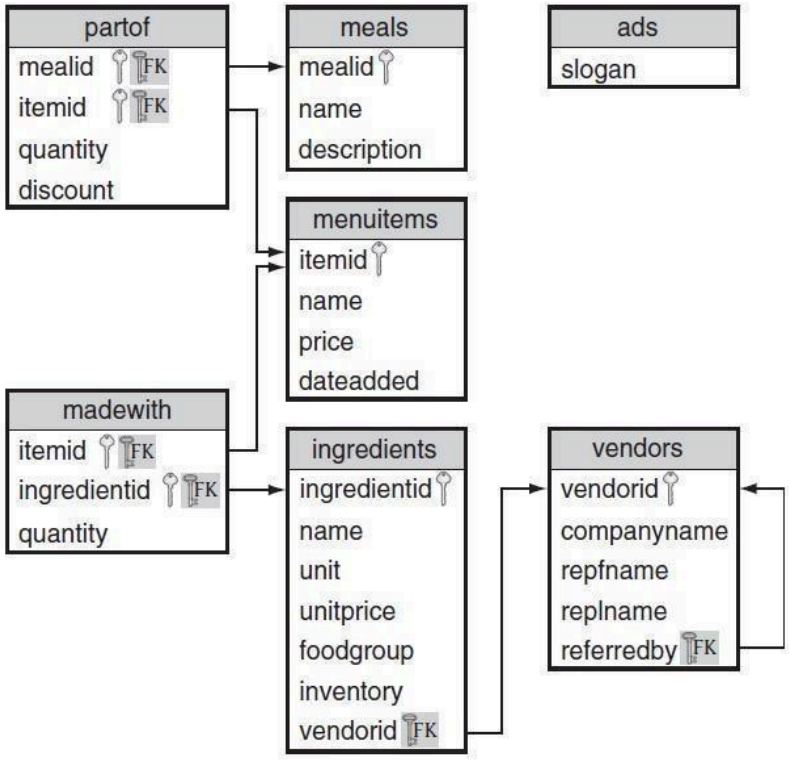
Birla Institute of Technology and Science, Pilani
CS F212 Database Systems
Lab No # 5

****

1 Sub-Queries

In this lab we will continue practicing SQL queries related to subqueries on a schema of a University created and populated in the previous labs. We also perform some examples on restaurant schema. The table schema for restaurant is given below.



So far, we've seen joins and set operators for combining tables together. SQL provides another way to combine tables. You can nest queries within queries. Such an embedded query is called a subquery. A subquery computes results that are then used by an outer query. Basically, a subquery acts like any other expression we've seen so far. A subquery can be nested inside the SELECT, FROM, WHERE, and HAVING clauses. You can even nest subqueries inside another subquery.

Example: Find store manager's mailing address of store 2.

```
select concat(address.address,', Dist. - ',address.district,'
PIN - ', address.postal_code) as address
```

```
from sakila.address
where address.address_id =
(select store.address_id from sakila.store
where store.manager_staff_id=2);
```

Some points to remember when using subqueries:

- SQL marks the boundaries of a subquery with parentheses.
- Only the columns of the outermost query can appear in the result table. When creating a new query, the outermost query must contain all the attributes needed in the answer.
- There must be some way of connecting the outer query to the inner query. All SQL comparison operators work with subqueries.
- Subqueries are restricted in what they can return. First, the row and column count must match the comparison operator. Second, the data types must be compatible.

1.1 Subqueries using IN

In the above query, the = operator makes the connection between the queries. Because = expects a single value, the inner query may only return a result with a single attribute and row. If subquery returns multiple rows, we must use a different operator. We use the IN operator, which expects a subquery result with zero or more rows.

Example 1: List all customer who done a payment with staff id =2.

```
select concat(customer.first_name,' ', customer.last_name) as
CustName
from sakila.customer
where customer.customer_id in
(select payment.customer_id from sakila.payment
where payment.staff_id=2);
```

Example 2: List name of student from student table who takes course in Fall semester.

```
select student.name from university2.student
where student.ID IN
(select takes.id from university2.takes
where takes.semester = 'Fall');
```

1.2 Subqueries Using NOT IN

IN over an empty table always returns false, therefore NOT IN always returns true.

Example: List name of student from student table who does not takes course in Fall semester.

```
select student.name from university2.student
where student.ID NOT IN
(select takes.id from university2.takes
where takes.semester = 'Fall');
```

1.3 Subqueries using BETWEEN

Subqueries can even be combined with other predicates in the WHERE clause, including other Subqueries. Look at the following query.

Example: Select instructor's name and ID whose salary is 5000 less or more than the average salary of all instructors in the university.

```
select instructor.ID, instructor.name
from university.instructor
where instructor.salary between
(select avg(instructor.salary)-5000
from university.instructor)
and
(select avg(instructor.salary)+5000
from university.instructor);
```

1.4 Subqueries with Empty Results

What happens when a subquery result is empty? The answer depends on what SQL is expecting from the subquery. Let's look at a couple of examples. Remember that the standard comparison operators expect a scalar value; therefore, SQL expects that any subqueries used with these operators always return a single value. If the subquery result is empty, SQL returns NULL.

```
select student.name from university.student
where student.ID =
(select takes.id from university.takes
where takes.semester = 'No Semester');
```

Here the subquery results are empty, so the subquery returns NULL. Evaluating the outer query, student.ID = NULL returns unknown for each row in vendors. Consequently, the outer query returns an empty result table.

When using the IN operator, SQL expects the subquery to return a table. When SQL expects a table from a subquery and the subquery result is empty, the subquery returns an empty table. IN over an empty table always returns false, therefore NOT IN always returns true.

```
select student.name from university.student
where student.ID NOT IN
(select takes.id from university.takes
```

```
where takes.semester = 'No Semester');
```

The above subquery returns a table with zero rows (and one column). For every row in the student table, NOT IN returns true over the subquery, so every row is returned.

2 More on Sub-Queries

2.1 Multilevel Subquery Nesting

Example: Find the list of course from course table, whose instructors getting more salary than the average salary of the university.

```
select course.course_id, course.title, course.credits
from university.course
where course.course_id in
(select teaches.course_id
from university.teaches, university.instructor
where teaches.ID in
(select avgsal.id from
(select instructor.id, instructor.salary
from university.instructor
having instructor.salary >
(select avg(instructor.salary) from university.instructor))
avgsal));
```

- GROUP BY, HAVING can be used in subqueries as well. Using ORDER BY in a subquery makes little sense because the order of the results is determined by the execution of the outer query.

2.2 Combining JOIN and Subqueries

- Nested queries are not restricted to a single table.

Example: Find the list of instructor's names, department, salary, section, semester, course title, and credits whose salary is more than the average salary of university.

```
select instructor.name, instructor.dept_name, instructor.salary,
teaches.sec_id, teaches.semester,
course.title, course.credits
from instructor natural join teaches
natural join course
having instructor.salary >
(select avg(instructor.salary) from university.instructor);
```

2.3 Standard Comparison Operators with Lists Using ANY, SOME, or ALL

We can modify the meaning of the SQL standard comparison operators with ANY, SOME, and ALL so that the operator applies to a list of values instead of a single value.

Using ANY or SOME:

- The ANY or SOME modifiers determine if the expression evaluates to true for at least one row in the subquery result.

Find all items that have a price that is greater than any salad item.

```
SELECT name
FROM items  WHERE price > ANY
(SELECT price
FROM items
WHERE name LIKE '%Salad');
```

Find all ingredients not supplied by Veggies_R_Us or Spring Water Supply

```
SELECT name
FROM ingredients
WHERE ingredientid NOT IN
(SELECT ingredientid
FROM ingredients  WHERE vendorid = ANY
(SELECT vendorid
FROM vendors
WHERE companyname = 'Veggies_R_Us' OR companyname
= 'Spring Water Supply'));
```

- Be aware of the difference between <> ANY and NOT IN. x <> ANY y returns true if any of the values in y are not equal to x. x NOT IN y returns true only if none of the values in y are equal to x or if the list y is empty.

Using ALL:

- The ALL modifier determines if the expression evaluates to true for all rows in the subquery result.

Find all ingredients that cost at least as much as every ingredient in a salad.

```
SELECT name
FROM ingredients
WHERE unitprice >= ALL
(SELECT unitprice
```

```
FROM ingredients ing JOIN madewith mw on
mw.ingredientid=ing.ingredientid JOIN
items i on mw.itemid=i.itemid WHERE i.name LIKE '%Salad');
```

Find the name of all ingredients supplied by someone other than Veggies_R_Us or Spring Water Supply.

```
SELECT name  FROM ingredients
WHERE vendorid <> ANY
(SELECT vendorid
FROM vendors
WHERE companyname = 'Veggies_R_Us' OR  companyname = 'Spring
Water Supply');
```

- Correct query:

```
SELECT name
FROM ingredients
WHERE ingredientid NOT IN (SELECT ingredientid FROM ingredients
WHERE vendorid = ANY
(SELECT vendorid FROM vendors  WHERE companyname =
'Veggies_R_Us' OR  companyname = 'Spring Water Supply'));
```

Find the most expensive items.

```
SELECT *
FROM items WHERE price >= ALL
(SELECT price FROM items);
```

What is wrong with above query? Correct query:

```
SELECT *
FROM items WHERE price >= ALL
(SELECT price FROM items where price is not null);
```

```
SELECT *
FROM items WHERE price >= ALL
(SELECT max(price) FROM items);
```

- You might be tempted to believe that >= ALL is equivalent to >= (SELECT MAX()), but this is not correct. What's going on here? Recall that for ALL to return true, the condition must be true for all rows in the subquery. NULL prices evaluate to unknown; therefore, >= ALL evaluates to unknown, so the result is empty. Of course,

we can solve this problem by eliminating NULL prices. However, NULL isn't the only problem. What happens when the subquery result is empty?

- A common mistake is to assume that the = ALL operator returns true if all the rows in the outer query match all of the rows in the inner query. This is not the case. A row in the outer query will satisfy the = ALL operator only if it is equal to all the values in the subquery. If the inner query returns multiple rows, each outer query row will only satisfy the = ALL predicate if all rows in the inner query the same value and that value have equals the outer query row value. Notice that the exact same result is achieved by using = alone and ensuring that only a distinct value is returned by the inner query.

2.4 Division Operation

One of the most challenging types of queries for SQL is one that compares two groups of rows to see if they are the same. These types of queries arise in many different applications. Some examples are as follows:

- Has a student taken all the courses required for graduation? o List the departments and projects for which that department has all the tools required for the project.
- Find all the items that contain all the ingredients provided by a vendor.

There are two approaches.

1. Using set operations
2. Using set cardinality.

- Cardinality of a relation is the number of tuples in that set.

Suppose A is set of customers and B is a set of payments. To check whether a customer done or not done a payment on sakila schema, compare two sets A and B.

Example 1: List all the customer who done any payment.

```
select * from customer
 where exists
(select * from payment
where payment.customer_id = customer.customer_id);
```

Example 2: List all the customer who not done any payment.

```
select * from customer
 where not exists
(select * from payment
where payment.customer_id = customer.customer_id);
```

- If a subquery returns any rows at all, EXISTS subquery is TRUE, and NOT EXISTS subquery is FALSE.

2.5 Comparing relational cardinality

Comparing relational cardinality: In this case we compare whether for every combination of A and B whether the number of elements in A and B are equal or not.

Find all items and vendors such that all ingredients in the item are supplied by that vendor.

```
SELECT i.name, companyname
FROM items i, vendors v
WHERE
(SELECT COUNT(DISTINCT m.ingredientid) -- number of ingredients
in item
FROM madewith m
WHERE i.itemid = m.itemid)
= -- number of ingredients in item supplied by vendor
(SELECT COUNT(DISTINCT m.ingredientid)
FROM madewith m, ingredients n
WHERE i.itemid = m.itemid AND m.ingredientid = n.ingredientid
AND
n.vendorid = v.vendorid);
```

Find all items and vendors such that all ingredients supplied by that vendor are in the item.

```
SELECT name, companyname FROM items i, vendors v
WHERE -- number of ingredients in item supplied by vendor
(SELECT COUNT(DISTINCT m.ingredientid)  FROM madewith m,
ingredients ing
WHERE i.itemid = m.itemid AND m.ingredientid = ing.ingredientid
AND ing.vendorid = v.vendorid)
=
(SELECT COUNT(DISTINCT ing.ingredientid) -- number of
ingredients supplied by vendor
FROM ingredients ing
WHERE ing.vendorid = v.vendorid);
```

2.6 Correlated Subqueries

Correlated subqueries are not independent of the outer query. Correlated subqueries work by first executing the outer query and then executing the inner query for each row from the outer query.

Find the name of students who took less than 10 courses.

```
select student.name from university.student
where
```

```
(select count(*) from takes
where student.ID = takes.ID)<10;
```

Look closely at the inner query. It cannot be executed independently from the outer query because the WHERE clause references the student table from the outer query. Note that in the inner query we must use the table name from the outer query to qualify ID. How does this execute? Because it's a correlated subquery, the outer query fetches all the rows from the student table. For each row from the outer query, the inner query is executed to determine the number of course for the particular ID.

EXISTS: EXISTS is a conditional that determines if any rows exist in the result of a subquery. EXISTS returns true if <subquery> returns at least one row and false otherwise.

Find the customer who do not done any payment at sakila schema.

```
select * from sakila.customer
where not exists
(select * from sakila.payment
where payment.customer_id = customer.customer_id);
```

2.7 Subqueries in the SELECT Clause

We can include subqueries in the SELECT clause to compute a column in the result table. It works much like any other expression. You may use both simple and correlated subqueries in the SELECT clause.

Example: Show the list of instructors in the university along with their ID, Dept Name, Salary, University's average salary, and difference from university's average salary from the instructor table.

```
select instructor.ID, instructor.name, instructor.salary,
(select avg(instructor.salary) from university.instructor) as
'AvgSalary',
(select avg(instructor.salary) from
university.instructor)-instructor.salary as 'Diff Salary'
from university.instructor;
```

- Correlated subqueries in the SELECT clause work just like they do in the WHERE clause.

2.8 Derived Relations — Subqueries in the FROM Clause

There are two advantages of derived relations. First, it allows us to break down complex queries into easier to understand parts. The second advantage of derived relations is that

we can improve the performance of some queries. If a derived relation is much smaller than the original relation, then the query may execute much faster.

Example: Show the list of instructors in the university along with their ID, Dept Name, Salary, Department's average salary from the instructor table.

```
select instructor.ID, instructor.name, instructor.dept_name,
instructor.salary, DeptAvgSal.AvgSal
from university.instructor,
(select instructor.dept_name, avg(instructor.salary) as 'AvgSal'
from university.instructor
group by instructor.dept_name) DeptAvgSal
where DeptAvgSal.dept_name = instructor.dept_name;
```

In the above query, select query generate a table with two fields, department name and departments average salary. This table to use to show the department's average salary corresponds to the department of instructor.

2.9 Subqueries in the HAVING Clause

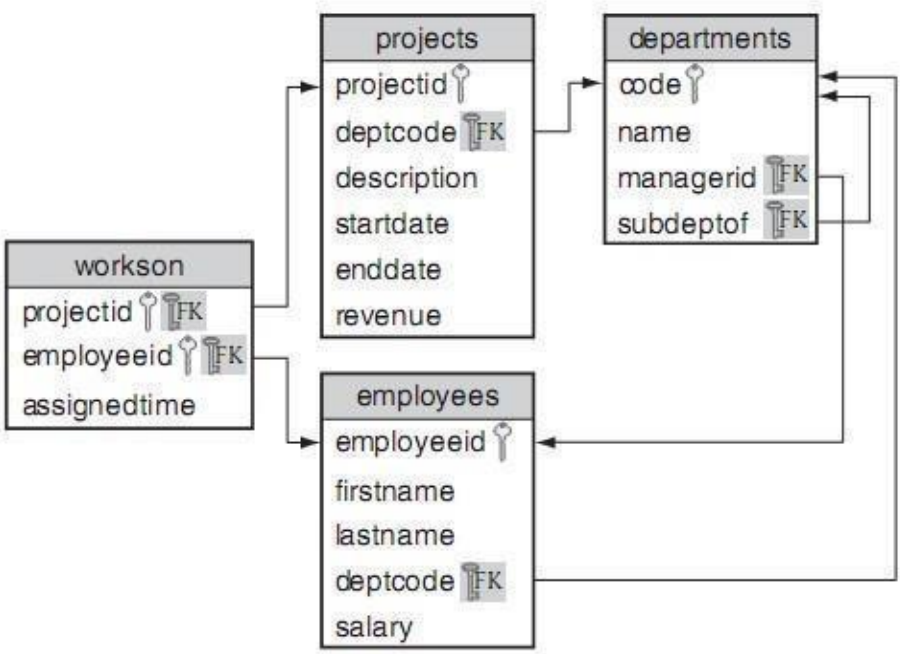
We can embed both simple and correlated subqueries in the HAVING clause. This works much like the WHERE clause subqueries, except we are defining predicates on groups rather than rows.

Example: Select the id, name, department name, salary and course id who gets salary more than the average salary of university.

```
select instructor.id, instructor.name, instructor.dept_name,
instructor.salary, teaches.course_id
from university.instructor, university.teaches
where instructor.id = teaches.id
having instructor.salary >
(select avg(instructor.salary) from university.instructor);
```

3 Exercises

Write a single SQL query for each of the following based on employee database. The scheme for employee database is shown below. Not all these queries can be solved using subqueries. Some of them require correlated subqueries and division operation.



1. Find the names of all people who work in the Consulting department.
2. Find the names of all people who work in the Consulting department and who spend more than 20% of their time on the project with ID ADT4MFIA.
3. Find the total percentage of time assigned to employee Abe Advice.
4. Find the names of all departments not currently assigned a project.
5. Find the first and last names of all employees who make more than the average salary of the people in the Accounting department.
6. Find the descriptions of all projects that require more than 70% of an employee's time.
7. Find the first and last name of all employees who are paid more than someone in the Accounting department.
8. Find the minimum salary of the employees who are paid more than everyone in the Accounting department.
9. Find the first and last name of the highest paid employee(s) in the Accounting department.
10. For each employee in the department with code ACCNT, find the employee ID and number of assigned hours that the employee is currently working on projects for other departments. Only report an employee if she has some current project to which she is assigned more than 50% of the time and the project is for another department. Report the results in ascending order by hours.
11. Find all departments where all their employees are assigned to all of their projects.

12. Use correlated subqueries in the SELECT and WHERE clauses, derived tables, and subqueries in the HAVING clause to answer these queries. If they cannot be answered using that technique, explain why.
    a. Find the names of all people who work in the Information Technology department.
    b. Find the names of all people who work in the Information Technology department and who spend more than 20% of their time on the health project.
    c. Find the names of all people who make more than the average salary of the people in the Accounting department.
    d. Find the names of all projects that require more than 50% of an employee's time. (e) Find the total percentage time assigned to employee Bob Smith.
    e. Find all departments did not assign a project.
    f. Find all employees who are paid more than someone in the Information Technology department.
    g. Find all employees who are paid more than everyone in the Information Technology department.
    h. Find the highest paid employee in the Information Technology department.

13. Fill-in-the-blanks:

Subquery Basics: A subquery can be nested inside the _____ clauses of a SQL query.

A) SELECT, FROM, WHERE, HAVING

B) INSERT, UPDATE, DELETE

C) CREATE, ALTER, DROP

D) TRUNCATE, MERGE, LOCK

Using Subqueries with Comparison Operators: When the inner subquery returns multiple rows, the outer query must use the _____ operator instead of the = operator to make a comparison.

A) >

B) <

C) IN

D) BETWEEN

Subqueries Using NOT IN: NOT IN always returns true when used over a/an _____ table.

A) empty

B) full

C) indexed

D) partitioned

Subqueries with Aggregate Functions: To select instructors whose salary is within 5000 less or more than the average salary, the subquery can use the _____ function combined with arithmetic operations.

A) SUM

B) COUNT

C) AVG

D) MAX

Correlated Subqueries: A correlated subquery is executed once for each row returned by the outer query. It references a column from a table in the outer query. This type of subquery is particularly useful for queries where you need to find rows in a table based on conditions relative to each row. For example, finding students who took less than 10 courses involves a correlated subquery counting courses per student. This statement is:

A) True

B) False

**********END**********