Database Systems and Web (15B11CI312)

Nested Subqueries

- A **subquery** is a **select-from-where** expression that is nested within another query.
- It is possible to embed a SQL statement within another.
- What is subquery useful for?
  - First, it can also be used to join tables.
  - Also, there are cases where the only way to correlate two tables is through a subquery.
- You can have more than one level of nesting in one single query.

- The syntax is as follows:

Outer Query

```
SELECT "column_name1"
FROM "table_name"
WHERE "column_name2" [Comparison Operator]
        (SELECT "column_name1"
        FROM "table_name“
        WHERE [Condition])
```

Sub-query/Inner Query

- [Comparison Operator] could be equality operators such as **=, >, <, >=, <=.**

  It can also be a text operator such as "LIKE."

Write a query that can returns name and enrolment number of CSE students.

Student(Enroll, SName, Age, DeptID, Address)
Department(DeptID, Name, HOD)

```
Select Enroll, SName
from Student
where DeptID IN( Select DeptID from
                 Department
                 where Name="CSE");
```

Outer Query

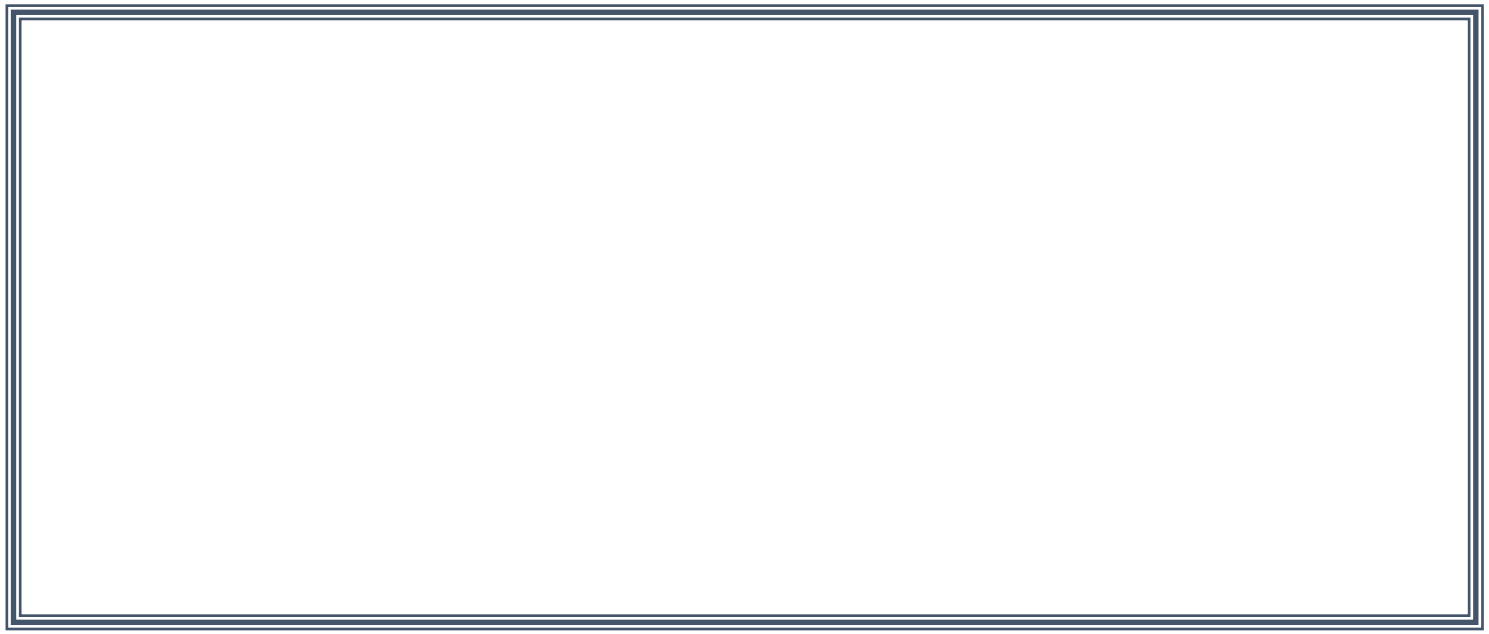
Inner Query

Step 1: The subquery returns the department ID of CSE department
Step 2: The outer query selects the enrolment number and name of students whose department code is in the result set returned from the subquery.

Construct (Operators)

- *The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query.*
- Constructs used in writing nested queries.
  - IN
  - NOT IN
  - ANY
  - SOME
  - ALL
  - EXIST
  - NOT-EXIST

“In” & “Not In” Construct



Find all customers who have both an account and a loan at the bank.

```
select distinct customer_name
    from borrower
    where customer_name in (select customer_name
                            from depositor );
```

Find all customers who have a loan at the bank but do not have an account at the bank.

```
select distinct customer_name
    from borrower
    where customer_name not in (select customer_name
                                from depositor );
```

Table ***Store_Information***

Store name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

Region_name	Store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Find the sales of all stores in the West region.

```
SELECT SUM(Sales) FROM Store_Information
    WHERE Store_name IN
    (SELECT store_name FROM Geography
        WHERE region_name = 'West')
```

SUM(Sales)
2050

In this example, instead of joining the two tables directly and then adding up only the sales amount for stores in the West region, we first use the subquery to find out which stores are in the West region, and then we sum up the sales amount for these stores.

“Some” or “Any” Construct

- The Some or Any operator returns true if any of the subquery values meet the condition.
- *The SOME and ANY comparison conditions do exactly the same thing and are completely interchangeable.*

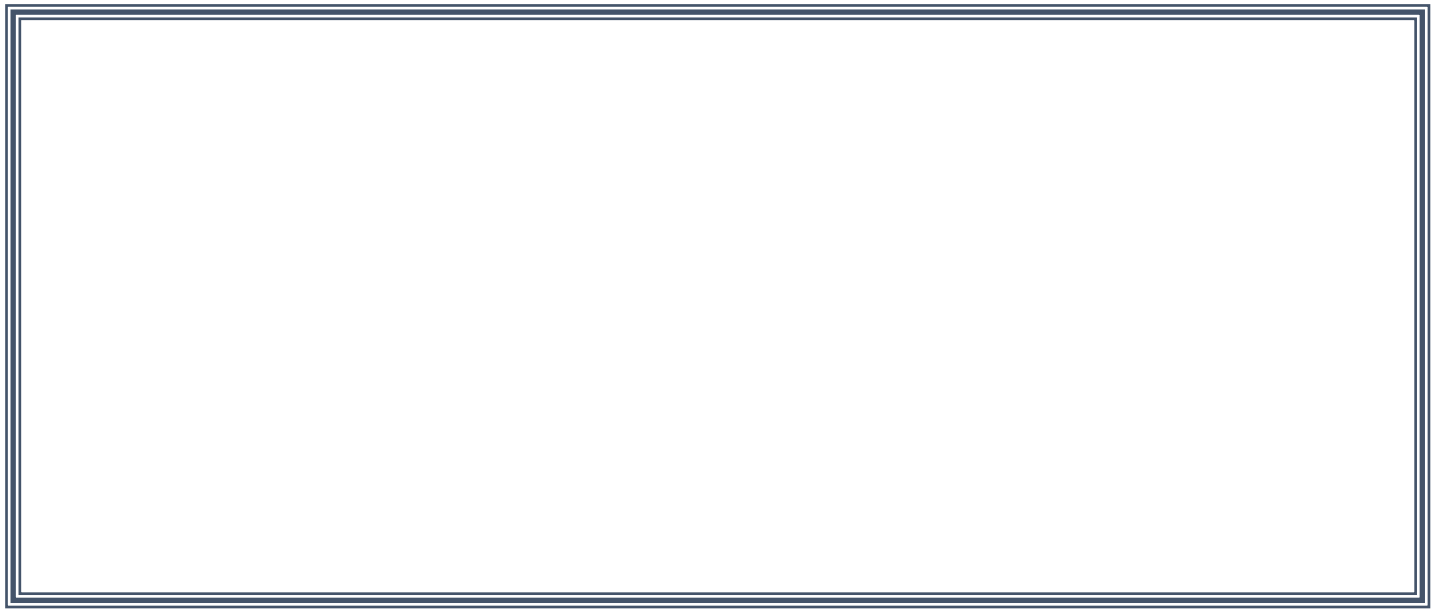
Syntax for SOME:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator SOME
(SELECT column_name FROM table_name
WHERE condition);
```

Syntax for ANY:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
(SELECT column_name FROM table_name
WHERE condition);
```

Find all branches that have greater assets than some branch located in Brooklyn.



- **Query using JOIN**

```
select distinct  T.branch_name
      from branch as T, branch as S
      where  T.assets > S.assets and
                  S.branch_city = 'Brooklyn';
```

- **Same query using Nested Query > some clause**

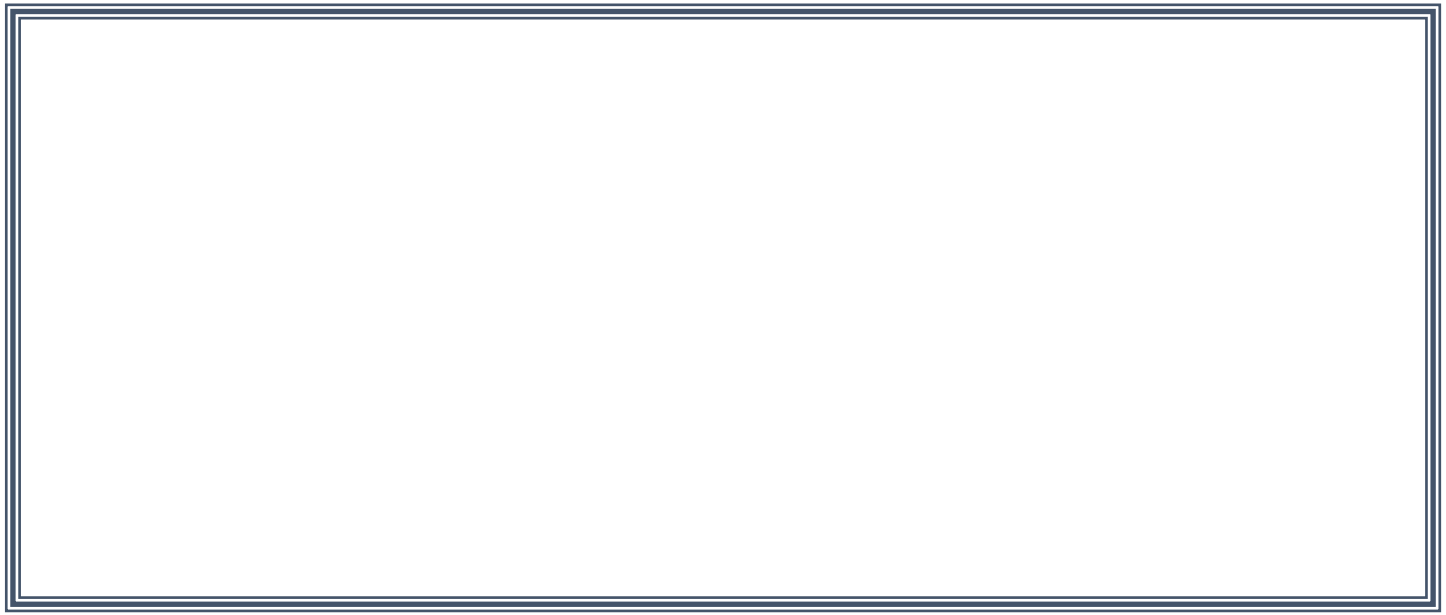
```
select branch_name
      from branch
      where assets > some
            (select assets
             from branch
             where branch_city = 'Brooklyn') ;
```

“All” Construct

The ALL operator returns true if all of the subquery values meet the condition

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
(SELECT column_name FROM table_name WHERE condition);
```



Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch_name
        from branch
        where assets > all
                (select assets
                from branch
                where branch_city = 'Brooklyn')
```

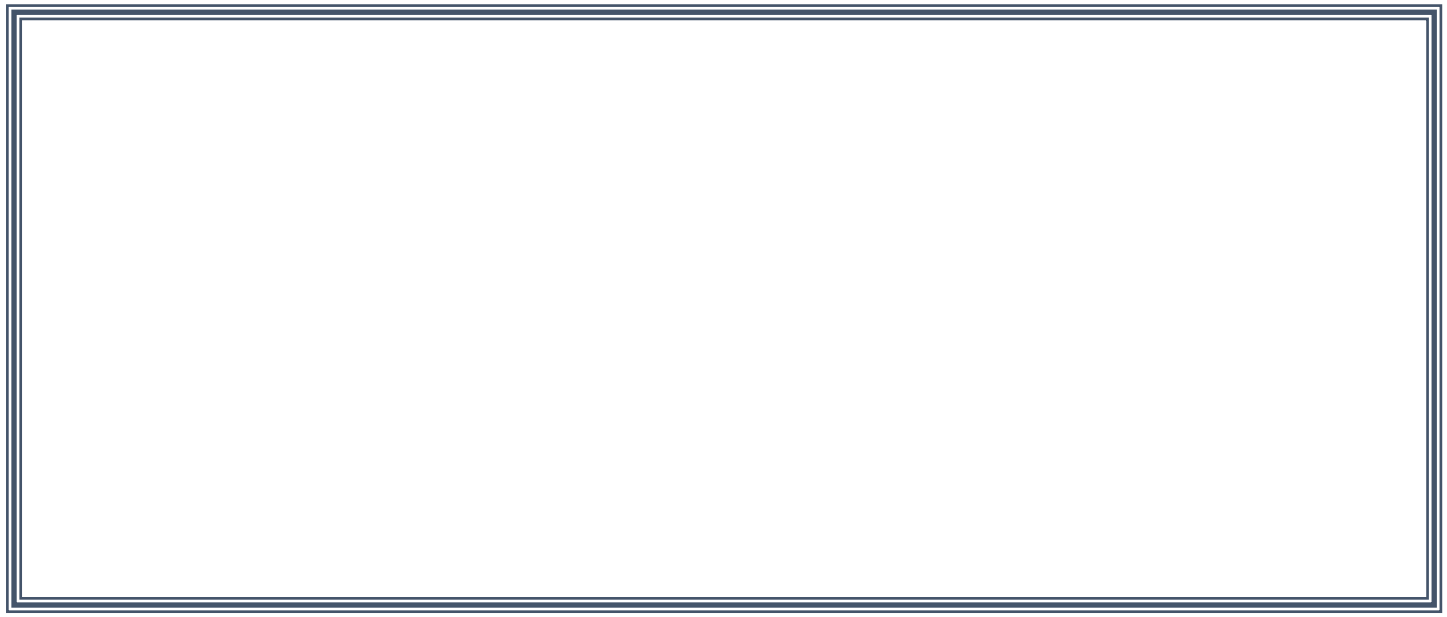
Exists & Not Exists construct

- When a subquery is used with **EXISTS or NOT EXISTS operator**, a subquery returns a Boolean value of TRUE or FALSE.
- The subquery acts as an existence check: SQL includes a feature for testing whether a subquery has any tuples in its result
- ***EXISTS*** *simply tests whether the inner query returns any row. If it does, then the outer query proceeds. If not, the outer query does not execute, and the entire SQL statement returns nothing.*

- The exists construct returns the value true if the argument subquery is nonempty.

Syntax for Exists

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```



Find all customers who have both an account and a loan at the bank.

```
select customer_name from borrower
    where EXISTS (select * from depositor where
    depositor.customer- name= borrower.customer-name);
```

Table ***Store_Information***

Store name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

Region_name	Store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Find the sales of all stores in the West region. To do so, we use the following SQL statement:

```
SELECT SUM(Sales) FROM Store_Information
WHERE EXISTS
(SELECT * FROM Geography
WHERE Region_Name = 'West');
```

SUM(Sales)

2750

At first, this may appear confusing, because the subquery includes the [region_name = 'West'] condition, yet the query summed up stores for all regions. Upon closer inspection, we find that since the subquery returns more than 0 row, the **EXISTS** condition is true, and the condition placed inside the inner query does not influence how the outer query is run.

The **NOT EXISTS** operator returns true if the subquery returns empty set of records i.e. there are no records in the result set of subquery.

However, if a single record is matched by the inner subquery, the **NOT EXISTS** operator will return false , and the subquery execution can be stopped.

Syntax for Not Exists

```
SELECT column_name(s)
FROM table_name
WHERE Not EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Sailor(sid, sname, rating, age)

SID	SNAME	RATING	AGE
18	A	3	30
41	B	6	56
22	C	7	44
63	D	NULL	15

Find the names of sailors with a higher rating than all sailors with age < 21.

```
SELECT S.sname
    FROM Sailors S
    WHERE NOT EXISTS ( SELECT *
                       FROM Sailors S2
                       WHERE S2.age < 21
                       AND S.rating <= S2.rating );
```

Find all customers who have an account at all the branches located in Brooklyn.

For each customer, we need to see whether the set of all branches at which that customer has an account contains the set of all branches in Brooklyn.

Using the except construct, we can write the query as follows:

```
select distinct S.customer-name from depositor as S where
 not exists ((select branch-name from branch
                        where branch-city = 'Brooklyn')
                       except
                        (select R.branch-name from depositor as T, account as R
                         where T.account-number = R.account-number
                          and S.customer-name = T.customer-name));
```

Here, the subquery

(select branch-name from branch

where branch-city = ’Brooklyn’) ;

finds all the branches in Brooklyn.

The subquery

(select R.branch-name from depositor as T, account as R

where T.account-number = R.account-number

and S.customer-name = T.customer-name)

finds all the branches at which customer S.customer-name has an account.

Thus, the outer select takes each customer and tests whether the set of all branches at which that customer has an account contains the set of all branches located in Brooklyn

Reading Suggestions for Exists & Not Exists

- https://dev.mysql.com/doc/refman/8.0/en/exists-and-not-exists-subqueries.html
- https://www.w3schools.com/sql/sql_exists.asp
- https://vladmihalcea.com/sql-exists/
- https://www.oracletutorial.com/oracle-basics/oracle-not-exists/
- https://www.red-gate.com/hub/product-learning/sql-prompt/consider-using-not-exists-instead-not-subquery

Sailors(sid: integer, sname: string, rating: integer, age: real)

Boats(bid: integer, bname: string, color: string)

Reserves(sid: integer, bid: integer, day: date)

Solve these queries using Nested queries.

1. Find the names of sailors who have reserved boat 103.
2. Find the name and the age of the youngest sailor.
3. Find the names and ratings of sailor whose rating is better than some sailor called Horatio
4. Find the names of sailors who have reserved all boats.
5. Find the names of sailors who have not reserved a boat whose name contains the string “storm”. Order the names in ascending order.
6. Find the name and the age of the youngest sailor.
7. Find the sailor id’s of sailors with the highest rating.
8. Find the sailor id’s of sailors whose rating is better than every sailor called Bob.
9. Find the name and age of the oldest sailor.

1. Find the names of sailors who have reserved boat 103.

Using IN	Using EXISTS
SELECT S.sname *FROM Sailors S* *WHERE S.sid IN (SELECT R.sid* *FROM Reserves R* *WHERE R.bid = 103) ;* The inner subquery has been completely independent of the outer query.	SELECT S.sname FROM Sailors S WHERE EXISTS (SELECT * FROM Reserves R WHERE R.bid = 103 AND R.sid = S.sid); The inner query depends on the row that is currently being examined in the outer query.

2. Find the name and the age of the youngest sailor.

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age <= ALL ( SELECT age
FROM Sailors ) ;
```

3. Find the names and ratings of sailor whose rating is better than some sailor called Harry.

```
SELECT S.sname, S.rating
FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
FROM Sailors S2
WHERE S2.sname = ‘Harry’ ) ;
```

4. Find the names of sailors who have reserved all boats.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS ( ( SELECT B.bid
                     FROM Boats B)
                                EXCEPT
                                        ( SELECT R.bid
                                          FROM Reserves R
                                          WHERE R.sid = S.sid ) );
```

5. **Find the names of sailors who have not reserved a boat whose name contains the string "storm". Order the names in ascending order**.

```
SELECT sname
FROM Sailors  s1
WHERE sid NOT IN
                    (SELECT sid
                    FROM r, s
                    WHERE r.sid=s.sid AND sname LIKE `%storm%')
                   ORDER BY s1.sname ;
```

6. Find the name and the age of the youngest sailor.

```
SELECT S.sname, S.age
FROM Sailors S
                WHERE S.age = (SELECT MIN(S2.age)
                 FROM Sailors S2 );
```

7. Find the sailor id's of sailors whose rating is better than every sailor called Bob.

```
SELECT sid
FROM Sailors s
WHERE rating > all (
                    SELECT rating
                    FROM Sailors s2
                    WHERE s2.sname=`Bob');
```

8. Find the sailor id's of sailors with the highest rating.

```
SELECT sid
FROM Sailors s1
WHERE s1.rating >= all (
                        SELECT rating
                        FROM Sailors s ) ;
```

9. Find the name and age of the oldest sailor.

```
SELECT s1.sname, s1.age
FROM sailors s1
WHERE s1.age >= all (
            SELECT age
            FROM sailors s) ;
```