CS 450

SQL - 2

Illustration of EXCEPT

Find the sids of all sailors who have reserved red boats ***but not*** *green boats:*

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color= ‘red’
EXCEPT
SELECT  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color= ‘green’ ;
```

Use MINUS instead of EXCEPT in Oracle

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all sailors whose ratings are null.

SELECT *S.sid*
FROM *Sailors S*
where *S.rating* **is null**

Nested Queries

- A **nested** query is a query that has another query embedded within it; this embedded query is called the **subquery**.
- Subqueries generally occur within the WHERE clause (but can also appear within the FROM and HAVING clauses)
- Nested queries are a very powerful feature of SQL. They help us write short and efficient queries.

(Think of nested **for** loops in C++. Nested queries in SQL are similar)

Nested Query 1

Find names of sailors who have reserved boat 103

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  ( SELECT  R.sid
                   FROM  Reserves R
                   WHERE  R.bid=103);
```

Nested Query 2

Find names of sailors who ***have not*** *reserved boat 103*

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid NOT IN  ( SELECT  R.sid
                       FROM  Reserves R
                       WHERE  R.bid=103 )
```

Nested Query 3

Find the names of sailors who have reserved a red boat

What about *Find the names of sailors who have NOT reserved a red boat?*

Revisit a previous query

Find names of sailors who've reserved a red ***and*** *a green boat*

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color= 'red'
INTERSECT
SELECT  S2.sid
FROM  Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
            AND B2.color= 'green' ;
```

Revisit a previous query

Find names of sailors who've reserved a red ***and*** *a green boat (using nested query)*

Correlated Nested Queries…1

- Thus far, we have seen nested queries where the inner subquery is independent of the outer query.
- We can make the inner subquery **depend** on the outer query. This is called <u>correlation</u>.

Correlated Nested Queries…2

Find names of sailors who have reserved boat 103

Tests whether the set is nonempty. If it is, then return TRUE.

```
SELECT  S.sname
FROM  Sailors S
WHERE  EXISTS  (SELECT  *
                FROM  Reserves R
                WHERE  R.bid=103 AND R.sid=S.sid);
```

(For finding sailors who have **not** reserved boat 103, we would use NOT EXISTS)

Correlated Nested Query - Division

Find the names of sailors who have reserved ALL boats (DIVISION)

```
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));
```

(For each sailor S, we check to see that the set of boats reserved by S includes every boat)

Correlated Nested Query 2

Alternatively,

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
                  WHERE NOT EXISTS (SELECT R.bid
                                    FROM Reserves R
                                    WHERE R.bid = B.bid AND
                                     R.sid = S.sid ));
```

NOT EXISTS vs. NOT IN

Employee

employee_id	employee_name	manager_id
1	John	5
2	David	5
3	Joe	5
4	Brandon	5
5	Chris	NULL
6	Jen	5
7	Kim	5
8	Mary	5
9	Dennis	5
10	Jim	5

NOT EXISTS vs. NOT IN

- Find the number of employees who are not managers

Try:

```
SELECT COUNT(*)
FROM Employee E
WHERE E.employee_id NOT IN
          (SELECT E2.manager_id
           FROM Employee E2);
```

NOT EXISTS vs. NOT IN

- Find the number of employees who are not managers

```
SELECT COUNT(*)
FROM Employee E
WHERE E.employee_id NOT IN
              (SELECT E2.manager_id
               FROM Employee E2);
```

COUNT = 0 (!)

NOT EXISTS vs. NOT IN

- Find the number of employees who are not managers

Try again:

```
SELECT COUNT(*)
FROM Employee E
WHERE NOT EXISTS
                (SELECT *
                  FROM Employee E2
                   WHERE E2.manager_id = E.employee_id);
```

NOT EXISTS vs. NOT IN

- Find the number of employees who are not managers

Try again:

```
SELECT COUNT(*)
FROM Employee E
WHERE NOT EXISTS
        (SELECT *
         FROM Employee E2
          WHERE E2.manager_id = E.employee_id);
```

COUNT = 9!

NOT EXISTS vs. NOT IN

- Find the number of employees who are not managers

Another option:

```
SELECT COUNT(*)
FROM Employee E LEFT OUTER JOIN Employee E2
        ON E.employee_id = E2.manager_id
WHERE E2.manager_id IS NULL;
```

NOT EXISTS vs. NOT IN

- Performance
  - NOT IN: Query performs nested full table scans
  - NOT EXISTS: Query can use an index within the sub-query.

UNIQUE operator

- When we apply UNIQUE to a subquery, it returns **true** if no row is duplicated in the answer to the subquery.
- What would the following SQL query return?

```
SELECT  S.sname
FROM  Sailors S
WHERE  UNIQUE  (SELECT  R.bid
                 FROM  Reserves R
                 WHERE  R.bid=103
                  AND R.sid=S.sid)
```

(All sailors with at most one reservation for boat 103.)

Note in Oracle, UNIQUE works like DISTINCT.

BETWEEN and AND operators

- The **BETWEEN** and **AND** operator selects a range of data between two values.
- These values can be numbers, text, or dates.

BETWEEN and **AND** Example

Find the names of sailors whose age is between 25 and 35

```
SELECT S.sname
FROM Sailors S
WHERE S.age BETWEEN 25 AND 35;
```

ANY/SOME, and ALL operators

Find sailors whose rating is better than some sailor named Horatio

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.sname= 'Horatio' );
```

Alternative is to use SOME, which is equivalent to ANY operator.

What if there are several sailors named Horatio?

Definition of "Any" (or "Some") Clause

F <comp> **any** $r \Leftrightarrow \exists\, t \in r$ such that (F <comp> t), where <comp> can be:
<, ≤, >, =, ≠

$(5 < \textbf{any} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$ (read: 5 < any tuple in the relation)

$(5 < \textbf{any} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{false}$

$(5 = \textbf{any} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$

$(5 \neq \textbf{any} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true (since } 0 \neq 5\text{)}$

(= **any**) ≡ **in**
However, (≠ **any**) ≢ **not in**

Substitute the "any" with "some", and you'll get the same result.

Using ALL operator

*Find sailors whose rating is better than **every** sailor named Horatio*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ALL(SELECT S2.rating
                     FROM Sailors S2
                      WHERE S2.sname= 'Horatio' );
```

Definition of All Clause

- F <comp> **all** $r \Leftrightarrow \forall\, t \in r$ (F <comp> t)

$$(5 < \textbf{all} \begin{array}{|c|}\hline 0\\\hline 5\\\hline 6\\\hline\end{array}) = \text{false}$$

$$(5 < \textbf{all} \begin{array}{|c|}\hline 6\\\hline 10\\\hline\end{array}) = \text{true}$$

$$(5 = \textbf{all} \begin{array}{|c|}\hline 4\\\hline 5\\\hline\end{array}) = \text{false}$$

$$(5 \neq \textbf{all} \begin{array}{|c|}\hline 4\\\hline 6\\\hline\end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

($\neq$ all) $\equiv$ not in
However, (= **all**) $\not\equiv$ **in**

Post Processing

- Processing on the result of an SQL query:
  - Sorting: can sort the tuples in the output by any column (even the ones not appearing in the SELECT clause)
  - Duplicate removal
  - Example:

```
SELECT  DISTINCT S.sname
FROM  Sailors S, Reserves R
WHERE  S.sid=R.sid AND R.bid=103
ORDER BY S.sid ASC, S.sname DESC;
```

- Aggregation operators

Aggregate operators

- What is aggregation?
  - Computing arithmetic expressions, such as **Minimum** or **Maximum**
- The aggregate operators supported by SQL are:

  COUNT, SUM, AVG, MIN, MAX

Aggregate Operators

- **COUNT**(A): The number of values in the column A
- **SUM**(A): The sum of all values in column A
- **AVG**(A): The average of all values in column A
- **MAX**(A): The maximum value in column A
- **MIN**(A): The minimum value in column A

(We can use DISTINCT with COUNT, SUM and AVG to compute only over non-duplicated columns)

Using the COUNT operator

Count the number of sailors

```
SELECT  COUNT (*)
FROM  Sailors S;
```

Example of SUM operator

Find the sum of ages of all sailors with a rating of 10

```
SELECT  SUM (S.age)
FROM  Sailors S
WHERE  S.rating=10;
```

Example of AVG operator

Find the average age of all sailors with rating 10

```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10;
```

Example of MAX operator

Find the name and age of the oldest sailor

```
SELECT S.sname, MAX(S.age)
FROM Sailors S;
```

But this is illegal in SQL!!

Correct SQL Query for MAX

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

Alternatively…

```
SELECT S.sname, S.age
FROM Sailors S
WHERE ROWNUM <= 1
ORDER BY S.age DESC;
```

Another Aggregate Query

Count the number of different sailor names

```
SELECT COUNT (DISTINCT S.sname)
FROM Sailors S
```