

SQL: Details on Subqueries



Refer to Head First SQL – chapter 9

NOTE

- We will be doing an in-class exercise to "discover" attributes of subqueries.
 - These notes are meant to be used after doing this exercise.
 - Solutions to all in-class exercises will be posted after class.
 - However, if you want to review these, it's fine.
-

Subquery Review

- Subqueries can be used:
 - In **SELECT clause** (scalar subquery)
 - In **FROM clause** (table subquery)
 - In **WHERE clause** (scalar or row subquery)
 - Subquery is always processed first. Result is “plugged into” outer query.
-

Subquery Review

- A Subquery may return
 - Scalar value – a single column from a single row
 - A Row – multiple columns from 1 row.
 - A Table – one or more columns from multiple rows.
 - Subquery will fail if it returns incorrect number of values for it's position in the outer query.
-

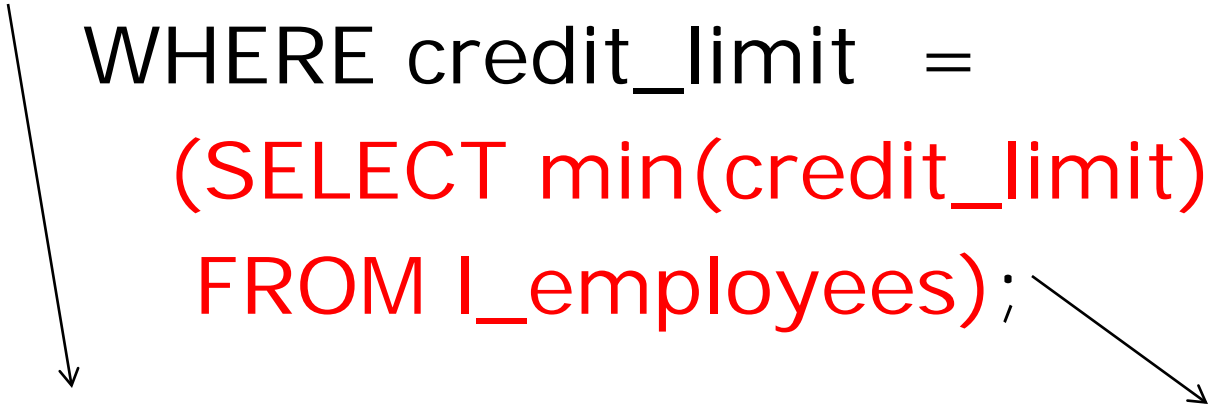
Scalar Subqueries

- Must return a **SINGLE value**.
 - Using count, sum, avg, max, min column function ensures that subquery will return a single value.
 - Condition in WHERE clause of outer query may use –
 - = (equal)
 - != (not equal)
 - < or <=
 - > or >=
 - IN, NOT IN
-

Scalar subquery example

```
SELECT last_name, credit_limit
FROM I_employees
WHERE credit_limit =
  (SELECT min(credit_limit)
   FROM I_employees);
```

*Find the last name for
all employees with
the minimum credit
limit.*



last_name	credit_limit
Owens	15
Jacobs	15

Credit_limit
15

Advice

- The subquery you just saw is the most common use for a subquery.
- Find some value in the database with a subquery. Then find all the records in the same table or a different table that have the value just returned by the subquery.
- Like answering two questions in a single query.
 - 1- What is the max credit limit?
 - 2 –Who has that max credit limit?

Try this: Find names of suppliers who have shipped part P2.

suppliers

sid	sname	status	city
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

shipments

sid	pid	qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

parts

pid	pname	color	weight	city
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris

Subquery with multiple values

- Find names of suppliers who have shipped part P2.
- Usually these subqueries are introduced with the 'IN' or 'NOT IN' conditions.

```
SELECT DISTINCT s.sname  
FROM sp_suppliers s  
WHERE s.sid IN
```

SNAME
CLARK
SMITH
BLAKE

```
(SELECT sp.sid  
FROM sp_shipments sp  
WHERE sp.pid = 'P2');
```

SID
S1
S3
S4

Find supplier names of P2. How could we do it as a Join?

SELECT

FROM

ON

WHERE

- Which is better? Subquery or Join?
 - Depends on your preference. (or brain?)
 - Some queries can be written as join or subquery; others cannot.
-

Nested Subquery

```
SELECT DISTINCT s.sname  
FROM suppliers s  
WHERE s.sid IN  
    (SELECT sp.sid  
    FROM shipments sp  
    WHERE sp.pid in  
        (SELECT p.pid  
        FROM parts p  
        WHERE p.color = 'RED'));
```

What is this query doing?

This subquery can also be rewritten as a join.

Find names of suppliers who have shipped Red Parts

suppliers

sid	sname	status	city
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

shipments

sid	pid	qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Step 3: get snames for S1,S2,S4

parts

pid	pname	color	weight	city
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris

Step 1: get pid for red parts

Step 2: get sid for P1,P4

Subqueries with IN, NOT IN

- What did we discover in our investigations about subqueries introduced by 'NOT IN'?
 - What can we do to fix the problem?
 - **!!!! Add another condition that excludes the NULLS!!!!**
-

Subquery that returns a table

- May be used in the FROM clause

```
SELECT s.sname, subquery1.total_qty  
FROM sp_suppliers s JOIN
```

```
(SELECT sid, sum(qty) AS total_qty  
FROM sp_shipments  
GROUP BY sid) subquery1
```

```
ON subquery1.sid = s.sid;
```

SNAME	TOTAL_QTY
BLAKE	200
CLARK	900
JONES	300
SMITH	1200

SID	TOTAL_QTY
S3	200
S4	900
S2	300
S1	1200

Subquery in the SELECT clause

- The subquery must return a single value.
- Example,

```
SELECT pid, qty, (SELECT city FROM  
                    sp_suppliers s  
                    WHERE s.sid = sp.sid)
```

```
FROM sp_shipments sp;
```

- Notice that the subquery is referencing the outer table (sp).
 - Is this allowed? YES.
-

Results:

PID	QTY	(SELECTCITYFROMSP_ SUPPLIERSWHERE S. SID=SP.SID)
P1	300	LONDON
P2	200	LONDON
P3	400	LONDON
P4	200	LONDON
P5	100	LONDON
P1	300	PARIS
P2	200	PARIS
P2	200	LONDON
P4	300	LONDON
P5	400	LONDON

Subqueries using EXISTS

- The EXISTS operator tests if the subquery returns at least one row.
 - The EXISTS operator returns either TRUE or FALSE, never unknown.
 - Because the EXISTS operator tests only if a row exists, the columns shown in the SELECT list are irrelevant.
 - Typically you use a text literal such as '1' or 'X'.
-

Subquery with Exists

- Find all suppliers who have shipped at least one part:

```
SELECT sname
```

```
FROM sp_suppliers s
```

```
WHERE EXISTS
```

```
  (SELECT 'X' FROM sp_shipments sh  
   WHERE sh.sid = s.sid);
```

```
SNAME
```

```
SMITH
```

```
JONES
```

```
BLAKE
```

```
CLARK
```

- This query says look at all the sids. If the sid appears in the sp_shipments table, print the sname.
 - Subquery with EXISTS is **ALWAYS correlated.**
-

Correlated Subqueries

- A correlated subquery contains references to tables included in the outer query.
 - This makes it difficult for the inner subquery to be processed first.
 - The DB must process the subquery for every row processed in the outer query.
 - Can be a *very costly* query!
-

Subquery with NOT Exists

- NOT EXISTS can also be used to test whether **no rows** are returned by the subquery.
- Find all suppliers who have not shipped any parts:

SELECT sname

FROM sp_suppliers s

WHERE **NOT EXISTS**

(SELECT 'X' FROM sp_shipments sh
WHERE sh.sid = s.sid);

SNAME
ADAMS

- This is also a correlated subquery
-

Correlated Subquery - Analysis

```
SELECT sname  
FROM sp_suppliers s  
WHERE EXISTS  
      (SELECT 'X' FROM sp_shipments sh  
       WHERE sh.sid = s.sid);
```

- The outer query knows nothing about the inner query except its results.
- Outer query cannot reference any columns in the subquery, but....
- Subquery has access to the outer query and can reference outer query columns.
- “You can look out, but you can’t look in”
- If there are 1,000 rows in shipments table,

how many times will subquery run?

Correlated Subquery Processing

- Select a row from the outer query.
 - Determine value of the correlated column(s)
 - For each record of the outer query, the inner query is executed.
 - The result of the inner query is then fed to the outer query and evaluated. If it satisfies the criteria, the row is returned for output
 - The next record of the outer query is selected and steps 2 through 4 are repeated until all the records of the outer query are evaluated.
-

Correlated Subquery

- Given the algorithm described on previous slide, what is it similar to?
 - Think about our French Database Consultant in the video?
 - Which join algorithm is the processing for a correlated subquery identical to?
 - Hash Join, Merge/Sort Join, or Nested Loops?
-

Not Exists v. Outer Join

- Before outer joins existed, had to simulate outer join with not exists query.

- Outer join:

```
SELECT sh.sid, s.sname  
FROM sp_suppliers s LEFT OUTER JOIN sp_shipments sh  
ON s.sid = sh.sid  
WHERE sh.sid IS NULL;
```

SID	SNAME
-	ADAMS

- NOT EXISTS subquery:

```
SELECT sname  
FROM sp_suppliers s  
WHERE NOT EXISTS  
  (SELECT 'X' FROM sp_shipments sh  
   WHERE sh.sid = s.sid);
```

SNAME
ADAMS

Nested Subqueries



- It is possible to nest subqueries 255 (!!) levels deep.
 - Just because you can doesn't mean you should
 - 2 or 3 layers of nesting is OK
-

Subqueries - Advice

- Try to avoid correlated subqueries.
 - Learn to read and understand them, but don't write them!
 - For queries that run often, test join performance v. subquery. There are situations where either could run faster.
-