CSCI 403
Database Management

10

Subqueries

IN

The IN Crowd

Another operator: IN

- Tests for presence in a list of values or tuples
- Can be used many places, but usually in WHERE
- Trivial examples:

```
SELECT 17 in (42, 99, 103);
 ?column?
----------
 f

SELECT 42 in (42, 99, 103);
 ?column?
----------
 t
```


IN in WHERE

```
SELECT course_id, section, instructor, title
FROM mines_courses
WHERE course_id IN
  ('CSCI261', 'CSCI262', 'CSCI999');
```

course_id	section	instructor	title
CSCI262	R01	Anderson, Alex	DATA STRUCTURES
CSCI262	R02	Painter-Wakefield, Christopher	DATA STRUCTURES
CSCI262	B	Painter-Wakefield, Christopher	DATA STRUCTURES
CSCI261	B	Gruchow, Colten	PROGRAMMING CONCEPTS
CSCI261	C	Paone, Jeffrey	PROGRAMMING CONCEPTS
CSCI261	D	Paone, Jeffrey	PROGRAMMING CONCEPTS
CSCI261	E	Schilling, Samuel	PROGRAMMING CONCEPTS
CSCI262	A	Anderson, Alex	DATA STRUCTURES
CSCI261	A	Gallegos, Lorenzo	PROGRAMMING CONCEPTS
CSCI262	R03	Painter-Wakefield, Christopher	DATA STRUCTURES

IN with Tuples

IN can also match *tuples* – ordered lists of values:

```
SELECT course_id, section, instructor, title
FROM mines_courses
WHERE (instructor, section) IN
 (('Fisher, Wendy','A'),
  ('Han, Qi','B'));
```

course_id	section	instructor	title
CSCI341	B	Han, Qi	COMPUTER ORGANIZATION
CSCI250	A	Fisher, Wendy	BUILDING A SENSOR SYSTEM
CSCI303	A	Fisher, Wendy	DATA SCIENCE

SUBQUERIES WITH IN

IN and Subqueries

Queries can return a list of values (or tuples).

We can substitute a query for a list:

```
SELECT * FROM mines_courses
WHERE instructor IN
  (SELECT name FROM mines_cs_faculty);
```

A *subquery*, also known as a *nested query*.

IN and Subqueries

```
SELECT * FROM mines_courses
WHERE instructor IN
  (SELECT name FROM mines_cs_faculty);
```

To interpret this, replace the subquery with the list resulting from the subquery:

('Han, Qi', 'Painter-Wakefield, Christopher', 'Paone, Jeffrey', …)

Another Example

Here we match *tuples*:

```
SELECT * FROM mines_courses
WHERE (instructor, section) IN
  (SELECT name, 'A' FROM mines_cs_faculty);
```

Of course, for this example we could instead do:

```
SELECT * FROM mines_courses
WHERE instructor IN
  (SELECT name FROM mines_cs_faculty)
AND section = 'A';
```


Subqueries vs Joins

Subquery queries are *often* equivalent to join queries:

```
SELECT * FROM mines_courses
WHERE instructor IN
  (SELECT name FROM mines_cs_faculty);
```

vs

```
SELECT mc.*
FROM mines_courses AS mc,
     mines_cs_faculty AS mcf
WHERE mc.instructor = mcf.name;
```


Subqueries vs Joins: A Difference

foo

x	y
apple	42
banana	17
cherry	99

bar

why	zee
42	2001-01-01
17	2002-02-02
42	2003-03-03

```
SELECT * FROM foo
WHERE y IN
  (SELECT why FROM bar);
```

x	y
apple	42
banana	17

```
SELECT foo.* FROM foo, bar
WHERE foo.y = bar.why;
```

x	y
apple	42
banana	17
apple	42

OTHER SUBQUERY SETTINGS

What Can a Subquery Return?

- A table (a list of tuples)
- A single tuple
- A (scalar) value
- Nothing

Subqueries Returning a Table

This is partly interpretation: all four cases fall under this category.

However, only some operators work with this general case:

[NOT] IN

[NOT] EXISTS

[NOT] UNIQUE (not implemented in PostgreSQL)

EXISTS & UNIQUE

Unlike IN, EXISTS and UNIQUE do not compare with expressions.

EXISTS returns true iff a subquery returns *anything*:

```
SELECT * FROM foo
WHERE EXISTS
    (SELECT * FROM bar WHERE why = 77);
```


EXISTS & UNIQUE

UNIQUE returns true iff a subquery returns unique values only.

Both of these operators are more useful in *correlated subqueries* (next section).

Subqueries Returning a Single Tuple

A query returning a scalar value is a special case of this.

A query returning nothing acts like a query returning a tuple of NULL values.

These subqueries can be used in equality comparison expressions (example next page):

Single Tuple Example

foo

x	y
apple	42
banana	17
cherry	99

bar

why	zee
42	2001-01-01
17	2002-02-02
42	2003-03-03

```
SELECT * FROM foo
WHERE y =
  (SELECT why FROM bar WHERE zee = '2002-02-02');

   x    | y
--------+----
 banana | 17
```

Single Tuple Counterexample

foo

x	y
apple	42
banana	17
cherry	99

bar

why	zee
42	2001-01-01
17	2002-02-02
42	2003-03-03

```
SELECT * FROM foo
WHERE y =
  (SELECT why FROM bar WHERE zee > '2001-01-01');
ERROR:  more than one row returned by a subquery
used as an expression
```


Non-Scalar Tuple

If you have multiple value tuples (not just a scalar as in previous example), use parentheses:

SELECT …
FROM *tablename*
WHERE (*expr1*, *expr2*, …) =
  (SELECT *sq_expr1*, *sq_expr2*, …);

Scalar Value

Scalar values can be used in any expression:

```
SELECT * FROM foo
WHERE y >
  (SELECT why FROM bar WHERE zee = '2002-02-02');
```

Doesn't have to be in WHERE, either:

```
SELECT
100 + (SELECT why FROM bar WHERE zee = '2002-02-02');
```

Again, a "nothing" result is interpreted as NULL.

CORRELATED SUBQUERIES

Correlated Subqueries 1

In a *correlated subquery*, the subquery accesses attributes from rows in the outer query.

Here's an example from Wikipedia:

```
SELECT employee_number, name
FROM employees AS e1
WHERE salary >
   (SELECT AVG(salary) FROM employees AS e2
    WHERE e2.department = e1.department);
```


Correlated Subqueries 2

```
SELECT AVG(salary) FROM employees AS e2
WHERE e2.department = e1.department
```

This subquery uses an operator we haven't covered yet, which provides an *aggregate* value over a table.

AVG(salary) gives the average of the salary values in all rows matching the WHERE condition.

Correlated Subqueries 3

```
SELECT employee_number, name
FROM employees AS e1
WHERE salary >
   (SELECT AVG(salary) FROM employees AS e2
    WHERE e2.department = e1.department);
```

The highlighted comparison shows the correlation.

Conceptually, a correlated subquery is run once *for every row* in the outer query.

The expression e1.department, then comes from some row in the outer query.

Correlated Subqueries 4

```
SELECT employee_number, name
FROM employees AS e1
WHERE salary >
   (SELECT AVG(salary) FROM employees AS e2
    WHERE e2.department = e1.department);
```

So what is this doing?

For each employee:

get the average of salaries in the employee's department;
if the employee's salary is greater than average, include it in the result

Another Example

Here's an example from the csci403 DB:

```
SELECT instructor, course id, section
FROM mines_courses AS mc1
WHERE course id IN
    (SELECT course id
    FROM mines_courses AS mc2
    WHERE mc2.course id = mc1.course id
    AND mc2.instructor <> mc1.instructor);
```


Another Example 2

The above query is equivalent to:

```
SELECT instructor, course id, section
FROM mines_courses AS mc1
WHERE EXISTS
    (SELECT course id
    FROM mines_courses AS mc2
    WHERE mc2.course id = mc1.course id
    AND mc2.instructor <> mc1.instructor);
```


SUBQUERIES IN OTHER CLAUSES

Subqueries in FROM

Not sure why you'd want to, but this is legal:

```
SELECT course_id FROM
    (SELECT course_id, instructor
    FROM mines_courses) AS mc
WHERE mc.instructor LIKE 'Painter%';
```

Subqueries in SELECT and SET

We saw an example using a subquery returning a scalar in a SELECT clause expression.

More usefully, we can use single-tuple subquery results in a SET clause in an UPDATE query (especially, with correlation).

This gives us something like a join that works with UPDATE!

Subquery in SET Example

From the PostgreSQL docs:

```
--Update contact names in an accounts table to
match the currently assigned salespeople:

UPDATE accounts SET (contact_first_name,
contact_last_name) =
   (SELECT first_name, last_name
    FROM salesperson
    WHERE salesperson.id = accounts.sales_id);
```


Up Next

- Next lecture:
  Grouping and aggregation.