Subqueries

Where can a subquery go?

- Relational algebra syntax is so elegant that it’s easy to see where subqueries can go.
- In SQL, a bit more thought is required . . .

Subqueries in a FROM clause

- In place of a relation name in the FROM clause, we can use a subquery.
- The subquery must be parenthesized.
- Must name the result, so you can refer to it in the outer query.

Worksheet, Q1:

```
SELECT sid, dept||cnum as course, grade
FROM Took,
   (SELECT *
    FROM Offering
    WHERE instructor='Horton') Hoffering
WHERE Took.oid = Hoffering.oid;
```

- This FROM is analogous to:

  $\text{Took} \times \rho_{\mathit{Hoffering}}\ (\mathit{«subquery»})$

- Can you suggest another version?

Subquery as a value in a WHERE

- If a subquery is guaranteed to produce exactly one tuple, then the subquery can be used as a value.
- Simplest situation: that one tuple has only one component.

Worksheet, Q2:

```
SELECT sid, surname
FROM Student
WHERE cgpa >
    (SELECT cgpa
     FROM Student
     WHERE sid = 99999);
```

- We can't do the analogous thing in RA:

$$\pi_{\text{sid, surname}}\ \sigma_{\text{cgpa} > (\textit{«subquery»})}\text{Student}$$

Special cases

- What if the subquery returns `NULL`?
- What if the subquery could return more than one value?

Quantifying over multiple results

- When a subquery can return multiple values, we can make comparisons using a quantifier.
- Example:

```
SELECT sid, surname
FROM Student
WHERE cgpa >
    (SELECT cgpa
     FROM Student
     WHERE campus = 'StG');
```

- We can require that
    - cgpa > **all** of them, or
    - cgpa > **at least one** of them.

The Operator ANY

- Syntax:

  x *«comparison»* ANY (*«subquery»*)

  or equivalently

  x *«comparison»* SOME (*«subquery»*)

- Semantics:

  Its value is true iff the comparison holds for at least one tuple in the subquery result, i.e.,

  ∃ y ∈ *«subquery results»* | x *«comparison»* y

- x can be a *list* of attributes,

  but this feature is not supported by psql.

The Operator ALL

- Syntax:

  x *«comparison»* ALL (*«subquery»*)

- Semantics:
  Its value is true iff the comparison holds for every tuple in the subquery result, i.e.,
  ∀ y ∈ *«subquery results»* | x *«comparison»* y

- x can be a list of attributes,
  but this feature is not supported by psql.

- Example: any-all

The Operator IN

- Syntax:
  x IN (*«subquery»*)
- Semantics:
  Its value is true iff x is in the set of rows generated by the subquery.
- x can be a list of attributes, and psql does support this feature.

Worksheet, Q3:

```
SELECT sid, dept||cnum AS course, grade
FROM Took NATURAL JOIN Offering
WHERE
   grade >= 80 AND
   (cnum, dept) IN (
         SELECT cnum, dept
         FROM Took NATURAL JOIN Offering
                   NATURAL JOIN Student
         WHERE surname = 'Lakemeyer');
```

UNIVERSITY OF TORONTO

Worksheet, Q4:

Suppose we have tables R(a, b) and S(b, c).

1. What does this query do?

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

2. Can we express this query without using IN?

The Operator EXISTS

- Syntax:
  EXISTS («*subquery*»)
- Semantics:
  Its value is true iff the subquery has at least one tuple.

- Read it as "exists a row in the subquery result"

Example: EXISTS

```
SELECT surname, cgpa
FROM Student
WHERE EXISTS (
   SELECT *
   FROM Took
   WHERE Student.sid = Took.sid and
         grade > 85 );
```

Worksheet, Q5:

```
SELECT instructor
FROM Offering Off1
WHERE NOT EXISTS (
   SELECT *
   FROM Offering
   WHERE
         oid <> Off1.oid AND
         instructor = Off1.instructor );
```

Worksheet, Q6:

```
SELECT DISTINCT oid
FROM Took
WHERE EXISTS (
    SELECT *
    FROM Took t, Offering o
    WHERE
        t.oid = o.oid AND
        t.oid <> Took.oid AND
        o.dept = ‘CSC' AND
        took.sid = t.sid );
```

Scope

- Queries are evaluated from the inside out.
- If a name might refer to more than one thing, use the most closely nested one.
- If a subquery refers only to names defined inside it, it can be evaluated once and used repeatedly in the outer query.
- If it refers to any name defined outside of itself, it must be evaluated once for each tuple in the outer query.
  These are called correlated subqueries.

Renaming can make scope explicit

```
SELECT instructor
FROM Offering Off1
WHERE NOT EXISTS (
  SELECT *
  FROM Offering Off2
  WHERE
    Off2.oid <> Off1.oid AND
    Off2.instructor = Off1.instructor );
```

Summary: where subqueries can go

- As a relation in a FROM clause.
- As a value in a WHERE clause.
- With ANY, ALL, IN or EXISTS in a WHERE clause.
- As operands to UNION, INTERSECT or EXCEPT.
- Reference: textbook, section 6.3.

Modifying a Database

Database Modifications

- Queries return a relation.
- A modification command does not; it changes the database in some way.
- Three kinds of modifications:
  - Insert a tuple or tuples.
  - Delete a tuple or tuples.
  - Update the value(s) of an existing tuple or tuples.

Two ways to insert

- We've already seen two ways to insert tuples into an empty table:

  `INSERT INTO` *«relation»* `VALUES` *«list of tuples»*`;`

  `INSERT INTO` *«relation»* `(`*«subquery»*`);`

- These can also be used to add tuples to a non-empty table.

Naming attributes in INSERT

- Sometimes we want to insert tuples, but we don’t have values for all attributes.
- If we name the attributes we *are* providing values for, the system will use `NULL` or a default for the rest.
- Convenient!

Example

```
CREATE TABLE Invite (
    name TEXT,
    campus TEXT DEFAULT 'StG',
    email TEXT,
    age INT);

INSERT INTO Invite(name, email)
(   SELECT firstname, email
    FROM Student
    WHERE cgpa > 3.4  );
```

Here, name and email get values from the query, campus gets the default value, and age gets `NULL`.

Deletion

- Delete tuples satisfying a condition:

```
DELETE FROM «relation»
WHERE «condition»;
```

- Delete all tuples:

```
DELETE FROM «relation»;
```

Example 1: Delete Some Tuples

```
DELETE FROM Course
WHERE NOT EXISTS (
   SELECT *
   FROM Took JOIN Offering
             ON Took.oid = Offering.oid
   WHERE
     grade > 50 AND
     Offering.dept = Course.dept AND
     Offering.cnum = Course.cnum
);
```

Updates

- To change the value of certain attributes in certain tuples to given values:

  `UPDATE` *«relation»*
  `SET` *«list of attribute assignments»*
  `WHERE` *«condition on tuples»*`;`

Example: update one tuple

- Updating one tuple:

```
UPDATE Student
SET campus = ‘UTM’
WHERE sid = 999999;
```

- Updating several tuples:

```
UPDATE Took
SET grade = 50
WHERE grade >= 47 and grade < 50;
```