Announcements

- Additional Office Hours and room changes
  - Website calendar is up-to-date
- Check email for Microsoft Azure invite
  “Action required: Accept your Azure lab assignment”

Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in:
  - A SELECT clause
    - Must return single value
  - A FROM clause
    - Can return multi-valued relation
  - A WHERE clause
- Rule of thumb: avoid nested queries when possible
  - But sometimes it’s impossible, as we will see

Subqueries in FROM

Sometimes we need to compute an intermediate table only to use it later in a SELECT-FROM-WHERE

- Option 1: use a subquery in the FROM clause
- Option 2: use the WITH clause
  - See textbook for details

Subqueries in WHERE

- SELECT ............ WHERE EXISTS (sub);
- SELECT ............ WHERE NOT EXISTS (sub);
- SELECT ............ WHERE attribute IN (sub);
- SELECT ............ WHERE attribute NOT IN (sub);
- SELECT ............ WHERE attribute > ANY (sub);
- SELECT ............ WHERE attribute > ALL (sub);
Monotone Queries

• Definition: A query \( Q \) is monotone if:
  – Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples.

Theorem: If \( Q \) is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

Proof. We use the nested loop semantics: if we insert a tuple in a relation \( R_i \), this will not remove any tuples from the answer.

The query:

\[
\text{SELECT } a_1, a_2, \ldots, a_k \text{ FROM } R_1 \text{ AS } x_1, R_2 \text{ AS } x_2, \ldots, R_n \text{ AS } x_n \\
\text{WHERE Conditions for } x_1 \text{ in } R_1 \text{ do } \\
\text{for } x_2 \text{ in } R_2 \text{ do } \\
\vdots \\
\text{for } x_n \text{ in } R_n \text{ do } \\
\text{if Conditions output } (a_1, \ldots, a_k)
\]

The query:

Find all companies s.t. all their products have price < 200 is not monotone.

Consequence: If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without nested subqueries.
Queries that must be nested

- Queries with universal quantifiers or with negation
- Queries that use aggregates in certain ways
  - \( \text{sum}(...) \) and \( \text{count}(*) \) are NOT monotone, because they do not satisfy set containment
  - \( \text{select count}(*) \) from \( R \) is not monotone!

SQL Idioms

Finding Witnesses

For each city, find the most expensive product made in that city

```
WITH CityMax AS
(SELECT x.city, max(y.price) as maxprice
FROM Company x, Product y
WHERE x.cid = y.cid
GROUP BY x.city;

SELECT u.city, v.pname, v.price
FROM Company u, Product v, CityMax w
WHERE u.cid = v.cid
  and u.city = w.city
  and v.price = w.maxprice;
```

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)
Finding Witnesses

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

```sql
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v,
(SELECT x.city, max(y.price) as maxprice
 WHERE x.cid = y.cid
 GROUP BY x.city) w
WHERE u.cid = v.cid
 and u.city = w.city
 and v.price = w.maxprice;
```

There is a more concise solution here:

```sql
SELECT u.city, v.pname, v.price
FROM Company u, Product v, Company x, Product y
WHERE u.cid = v.cid and u.city = x.city
 and x.cid = y.cid
GROUP BY u.city, v.pname, v.price
HAVING v.price = max(y.price);
```

SQL: Our first language for the relational model

- Projections
- Selections
- Joins (inner and outer)
- Inserts, updates, and deletes
- Aggregates
- Grouping
- Ordering
- Nested queries

Relational Algebra

- Set-at-a-time algebra, which manipulates relations
- In SQL we say what we want
- In RA we can express how to get it
- Every DBMS implementation converts a SQL query to RA in order to execute it
- An RA expression is called a query plan

Why study another relational query language?

- RA is how SQL is implemented in DBMS
  - We will see more of this in a few weeks
- RA opens up opportunities for query optimization
Basics

• Relations and attributes
• Functions that are applied to relations
  – Return relations
    \( R_2 = \sigma_{c}(R_1) \)
  – Can be composed together
    \( R_3 = \pi_{c}(R_1) \)
  – Often displayed using a tree rather than linearly
  – Use Greek symbols: \( \sigma, \pi, \delta, \) etc

Sets v.s. Bags

• Sets: \( \{a, b, c\}, \{a, d, e, f\}, \{\} \ldots \)
• Bags: \( \{a, a, b, c\}, \{b, b, b, b\}, \ldots \)

Relational Algebra has two flavors:
• Set semantics = standard Relational Algebra
• Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)

Relational Algebra Operators

• Union \( \cup \), intersection \( \cap \), difference
• Selection \( \sigma \)
• Projection \( \pi \)
• Cartesian product \( \times \), join \( \Join \)
• (Rename \( \rho \))
• Duplicate elimination \( \delta \)
• Grouping and aggregation \( \gamma \)
• Sorting \( \# \)

All operators take in 1 or more relations as inputs and return another relation

Union and Difference

\[ R_1 \cup R_2 \]
\[ R_1 - R_2 \]

Only make sense if \( R_1, R_2 \) have the same schema

What do they mean over bags?

What about Intersection?

• Derived operator using minus
  \[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]

• Derived using join
  \[ R_1 \cap R_2 = R_1 \Join R_2 \]

Selection

• Returns all tuples which satisfy a condition
  \[ \sigma_c(R) \]

• Examples
  – \( \sigma_{\text{Salary} > 4000}(\text{Employee}) \)
  – \( \sigma_{\text{Name} = \text{"Smith"}}(\text{Employee}) \)
  – The condition \( c \) can be \( =, \ <, \ <\leq, \ >, \ \geq, \ <> \) combined with AND, OR, NOT
Projection

- Eliminates columns
  \[ \pi_{A_1, \ldots, A_n}(R) \]

- Example: project social-security number and names:
  \[ \pi_{\text{SSN}, \text{Name}}(\text{Employee}) \rightarrow \text{Answer(\text{SSN}, \text{Name})} \]

Different semantics over sets or bags! Why?

Composing RA Operators

Cartesian Product

- Each tuple in R1 with each tuple in R2
  \[ R_1 \times R_2 \]

- Rare in practice; mainly used to express joins
**Renaming**

- Changes the schema, not the instance
  \[ \rho_{B_1 \ldots B_n}(R) \]
- Example:
  - Given `Employee(Name, SSN)`
  - `\rho_{N, S}(Employee) \rightarrow Answer(N, S)`

**Natural Join**

- **R1 \Join R2**
- Meaning: `R1 \Join R2 = \Pi_A(\sigma_0 (R1 \times R2))`
- Where:
  - Selection `\sigma_0` checks equality of all common attributes (i.e., attributes with same names)
  - Projection `\Pi_A` eliminates duplicate common attributes

### Natural Join Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

### Natural Join Example 2

<table>
<thead>
<tr>
<th>AnonPatient P</th>
<th>Voters V</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>zip</td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

### Theta Join

- A join that involves a predicate
  \[ R1 \bowtie_\theta R2 = \sigma_0 (R1 \times R2) \]
- Here `\theta` can be any condition
- No projection in this case!
- For our voters/patients example:
  \[ P \bowtie_\theta p.zip = v.zip \land p.Age < 1 \land p.Age > v.Age \]

CSE 414 - Autumn 2018
Equijoin

• A theta join where θ is an equality predicate

\[ R_1 \bowtie_\theta R_2 = \sigma_\theta (R_1 \times R_2) \]

• By far the most used variant of join in practice
• What is the relationship with natural join?

Join Summary

• **Theta-join**: \( R \bowtie_\theta S = \sigma_\theta (R \times S) \)
  – Join of R and S with a join condition \( \theta \)
  – Cross-product followed by selection \( \theta \)
  – No projection
• **Equijoin**: \( R \bowtie_\theta S = \sigma_\theta (R \times S) \)
  – Join condition \( \theta \) consists only of equalities
  – No projection
• **Natural join**: \( R \bowtie S = \pi_A (\sigma_\theta (R \times S)) \)
  – Equality on all fields with same name in R and in S
  – Projection \( \pi_A \) drops all redundant attributes

So Which Join Is It?

When we write \( R \bowtie S \) we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context

More Joins

• **Outer join**
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes
  – Does not eliminate duplicate columns

• Variants
  – Left outer join
  – Right outer join
  – Full outer join

Outer Join Example
Some Examples

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)

Name of supplier of parts with size greater than 10
Project(sname)[Supplier Join(sno=sno)
(Supply Join(pno=pno) (Select( psize>10)(Part)))]

Using symbols:
πsname(Supplier ⨝ Supply ⨝ (σpsize>10(Part)))

Can be represented as trees as well

Representing RA Queries as Trees

Name of supplier of parts with size greater than 10
Project(sname)[Supplier Join(sno=sno)
(Supply Join(pno=pno) (Select( psize>10)(Part)))]

Using symbols:
πsname(Supplier ⨝ Supply ⨝ (σpsize>10(Part)))

Can be represented as trees as well

Relational Algebra Operators

- Union ∪
- Intersection ∩
- Difference
- Selection σ
- Projection π
- Cartesian product X, join ⨝
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

All operators take in 1 or more relations as inputs and return another relation

Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
  - Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
- Sorting τ
  - Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.
Using Extended RA Operators

Typical Plan for a Query (1/2)

Typical Plan for a Query (1/2)

How about Subqueries?

How about Subqueries?

How about Subqueries?

Typical Plan for a Query (1/2)
How about Subqueries?

**SELECT Q.sno**
**FROM Supplier Q**
**WHERE Q.sstate = 'WA'**
  **and not exists**
  **(SELECT * FROM Supply P**
  **WHERE P.sno = Q.sno**
  **and P.price > 100)**

De-Correlation

**SELECT Q.sno**
**FROM Supplier Q**
**WHERE Q.sstate = 'WA'**
**EXCEPT**
**SELECT P.sno**
**FROM Supply P**
**WHERE P.price > 100**

Un-nesting

How about Subqueries?

**SELECT Q.sno**
**FROM Supplier Q**
**WHERE Q.sstate = 'WA'**
**EXCEPT**
**SELECT P.sno**
**FROM Supply P**
**WHERE P.price > 100**

EXCEPT = set difference

Summary of RA and SQL

- SQL = a declarative language where we say *what* data we want to retrieve
- RA = an algebra where we say *how* we want to retrieve the data
- **Theorem**: SQL and RA can express exactly the same class of queries

RDBMS translates SQL to RA, then optimizes RA

Summary of RA and SQL

- SQL (and RA) cannot express ALL queries that we could write in, say, Java
- Example:
  - Parent(p,c): find all descendants of 'Alice'
  - No RA query can compute this!
  - This is called a recursive query
- Next lecture: Datalog is an extension that can compute recursive queries