Announcements

- HW3 is out – due Friday
  - `git pull upstream master`
  - Make sure you have email from Microsoft Azure and log in
- Web quiz 2 due tonight

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*
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 \( U \), intersection \( \cap \), difference \( - \)
- Selection \( \sigma \)
- Projection \( \pi \)
- Cartesian product \( \times \), join \( \Join \)
- (Rename \( \rho \))
- Duplicate elimination \( \delta \)
- Grouping and aggregation \( \gamma \)
- Sorting \( \tau \)

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

Union and Difference

\[ R1 \cup R2 \\
R1 - R2 \]

Only make sense if \( R1, R2 \) have the same schema

What do they mean over bags?

Selection

- Returns all tuples which satisfy a condition \( \sigma_c(R) \)
- Examples
  - \( \sigma_{\text{Salary} > 40000}(\text{Employee}) \)
  - \( \sigma_{\text{Name} = \text{"Smith"}}(\text{Employee}) \)
- The condition \( c \) can be \( =, <, \leq, >, \geq, <> \) combined with AND, OR, NOT

What about Intersection?

- Derived operator using minus
  \[ R1 \cap R2 = R1 - (R1 - R2) \]
- Derived using join
  \[ R1 \cap R2 = R1 \Join R2 \]

Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>

\( \sigma_{\text{Salary} = 60000}(\text{Employee}) \)

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>
Projection

- Eliminates columns

$$\pi_{A_1, \ldots, A_n}(R)$$

- Example: project social-security number and names:

  $$\pi_{\text{SSN}, \text{Name}}(\text{Employee}) \rightarrow \text{Answer(} \text{SSN, Name}\text{)}$$

Different semantics over sets or bags! Why?

Composing RA Operators

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>flu</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

$$\sigma_{\text{disease} = \text{heart}}(\text{Patient})$$

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

$$\pi_{\text{zip}, \text{disease}}(\sigma_{\text{disease} = \text{heart}}(\text{Patient}))$$

Cartesian Product

- Each tuple in R1 with each tuple in R2

$$R_1 \times R_2$$

- Rare in practice; mainly used to express joins

Cross-Product Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

Renaming

- Changes the schema, not the instance

$$\rho_{B_1, \ldots, B_n}(R)$$

- Example:

  - Given Employee(Name, SSN)
  - $$\rho_N, S(\text{Employee}) \rightarrow \text{Answer(N, S)}$$
Natural Join

\[ R_1 \bowtie R_2 \]

• Meaning: \( R_1 \bowtie R_2 = \Pi_A(\sigma_q(R_1 \times R_2)) \)

• Where:
  – Selection \( \sigma_q \) 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>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

\( R \bowtie S = \Pi_{B,C,E}(\sigma_{R.B=S.B}(R \times S)) \)

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>

<table>
<thead>
<tr>
<th>P \bowtie V</th>
<th></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>

Natural Join

• Given schemas \( R(A, B, C, D), S(A, C, E) \), what is the schema of \( R \bowtie S \)?

• Given \( R(A, B, C), S(D, E) \), what is \( R \bowtie S \)?

• Given \( R(A, B), S(A, B) \), what is \( R \bowtie S \)?

Theta Join

• A join that involves a predicate

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

• Here \( \theta \) can be any condition

• No projection in this case!

• For our voters/patients example:

\[ P \bowtie \lambda P.\text{zip} = V.\text{zip} \text{ and } P.\text{age} \geq V.\text{age} \text{ and } P.\text{age} \leq V.\text{age} + 1 \]

Equijoin

• A theta join where \( \theta \) 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?
Equijoin Example

AnonPatient $P$

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

Voters $V$

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

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

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

$\pi_{sname}(\text{Supplier Join}(\text{Supplier Join}(\pi_s(sno) \bowtie \text{Supply}(sno, pno, qty, price))))$

Using symbols:

$\pi_{sname}(\text{Supplier Join}(\text{Supply Join}(\text{Supplier Join}(\pi_p(pno) \bowtie \text{Part}))))$

Can be represented as trees as well
Representing RA Queries as Trees

Some Examples

Relational Algebra Operators

Extended RA: Operators on Bags

Grouping

Using Extended RA Operators
Typical Plan for a Query (1/2)

Answer

\[\pi\text{fields} \quad \sigma\text{selection condition} \quad \text{SELECT PROJECT JOIN} \quad \text{Query}\]

R  S

Typical Plan for a Query (1/2)

\[\sigma\text{having condition} \quad \gamma\text{fields, sum/count/min/max(fields)} \quad \text{SELECT fields} \quad \text{FROM R, S, -- \ WHERE condition} \quad \text{GROUP BY fields} \quad \text{HAVING condition}\]

CSE 414 - Autumn 2018

How about Subqueries?

Return all suppliers in WA that sell no products greater than $100

CSE 414 - Autumn 2018

How about Subqueries?

\[
\text{SELECT Q.sno} \\
\text{FROM Supplier Q} \\
\text{WHERE Q.sstate = 'WA' and not exists (SELECT * FROM Supply P WHERE P.sno = Q.sno and P.price > 100)}
\]

CSE 414 - Autumn 2018

How about Subqueries?

Option 1: create nested plans

CSE 414 - Autumn 2018

How about Subqueries?

\[
\text{SELECT Q.sno} \\
\text{FROM Supplier Q} \\
\text{WHERE Q.sstate = 'WA' and not exists (SELECT * FROM Supply P WHERE P.sno = Q.sno and P.price > 100)}
\]

Correlation!
How about Subqueries?

```sql
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

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

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 translate SQL \(\rightarrow\) RA, then optimize 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

Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
  - Data models, SQL, Relational Algebra, Datalog
- Unit 3: Non-relational data
- Unit 4: RDBMS internals and query optimization
- Unit 5: Parallel query processing
- Unit 6: DBMS usability, conceptual design
- Unit 7: Transactions
What is Datalog?

• Another query language for relational model
  – Designed in the 80’s
  – Simple, concise, elegant
  – Extends relational queries with recursion

• Today is a hot topic:
  – Souffle (we will use in HW4)
  – Eve http://witheve.com/
  – Differential datalog
    https://github.com/frankmcsherry/differential-dataflow
  – Beyond databases in many research projects:
    network protocols, static program analysis

Example: storing FB friends

Why bother with yet another relational query language?

Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

When does it end???
**Datalog: Facts and Rules**

**Facts** = tuples in the database

```
Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)
```

**Rules** = queries

```
decl Actor(id:number, fname:symbol, lname:symbol)
decl Casts(pid: number, mid: number)
decl Movie(id:number, name:symbol, year:number)
```

**Insert data**

- `Actor(344759, 'Douglas', 'Fowley')`
- `Casts(344759, 29851)`
- `Casts(355713, 29000)`
- `Movie(7909, 'A Night in Armour', 1910)`
- `Movie(29000, 'Arizona', 1940)`
- `Movie(29445, 'Ave Maria', 1940)`

**Table declaration**

Types in Souffle: number (aka varchar)

**Types in Souffle: symbol (aka varchar)**

**Find Movies made in 1940**

```
Q1(y) : - Movie(x, y, z), z=1940.
```

**Order of variable matters!**

```
SELECT name FROM Movie WHERE year = 1940
```
Datalog: Facts and Rules

**Facts** = tuples in the database

**Rules** = queries

Find Movies made in 1940

Find Actors who acted in Movies made in 1940

Find Actors who acted in a Movie in 1940 and in one in 1910