CMSC424: Database Design
Relational Model; SQL

February 5, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu
Today’s Plan

- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)

- Relational Algebra Continued
  - Different types of joins
  - Formal semantics of SQL

- SQL Continued (if time)
  - Null values (3.6)
  - Aggregates (3.7)
  - Also the focus of next reading assignment
Find the names of all instructors:

```
select name
from instructor
```

Select all attributes:

```
select *
from instructor
```

Expressions in the select clause:

```
select name, salary < 100000
from instructor
```

More complex filters:

```
select name
from instructor
where (dept_name != 'Finance' and salary > 75000)
or (dept_name = 'Finance' and salary > 85000);
```

A filter with a subquery:

```
select name
from instructor
where dept_name in (select dept_name from department where budget < 100000);
```
Find the names of all instructors:

```sql
select name
from instructor
```

More complex expressions:

```sql
select concat(name, concat(' ', dept_name))
from instructor;
```

Careful with NULLs:

```sql
select name
from instructor
where salary < 100000 or salary >= 100000;
```

Wouldn’t return the instructor with NULL salary (if any)

Renaming tables or output column names:

```sql
select i.name, i.salary * 2 as double_salary
from instructor i
where i.salary < 80000 and i.name like '%g_';
```
Multi-table Queries

Use predicates to only select “matching” pairs:

```
select *
from instructor i, department d
where i.dept_name = d.dept_name;
```

Cartesian product:

```
select *
from instructor, department
```

Identical (in this case) to using a natural join:

```
select *
from instructor natural join department;
```

Natural join does an equality on common attributes – doesn’t work here:

```
select *
from instructor natural join advisor;
```

Instead can use “on” construct (or where clause as above):

```
select *
from instructor join advisor on (i_id = id);
```
Multi-table Queries

3-Table Query to get a list of instructor-teaches-course information:

```sql
select i.name as instructor_name, c.title as course_name
from instructor i, course c, teaches
where i.ID = teaches.ID and c.course_id = teaches.course_id;
```

Beware of unintended common names (happens often)
You may think the following query has the same result as above – it doesn’t

```sql
select name, title
from instructor natural join course natural join teaches;
```

I prefer avoiding “natural joins” for that reason

Note: On the small dataset, the above two have the same answer, but not on the large dataset. Large dataset has cases where an instructor teaches a course from a different department.
Set operations

Find courses that ran in Fall 2009 or Spring 2010

\[
\text{(select course_id from section where semester = 'Fall' and year = 2009)}
\]
\[
\text{union}
\]
\[
\text{(select course_id from section where semester = 'Spring' and year = 2010)}
\]

In both:

\[
\text{(select course_id from section where semester = 'Fall' and year = 2009)}
\]
\[
\text{intersect}
\]
\[
\text{(select course_id from section where semester = 'Spring' and year = 2010)}
\]

In Fall 2009, but not in Spring 2010:

\[
\text{(select course_id from section where semester = 'Fall' and year = 2009)}
\]
\[
\text{except}
\]
\[
\text{(select course_id from section where semester = 'Spring' and year = 2010)}
\]
Set operations: Duplicates

Union/Intersection/Except eliminate duplicates in the answer (the other SQL commands don’t) (e.g., try ‘select dept_name from instructor’).

Can use “union all” to retain duplicates.

NOTE: The duplicates are retained in a systematic fashion (for all SQL operations)

Suppose a tuple occurs $m$ times in $r$ and $n$ times in $s$, then, it occurs:

- $m + n$ times in $r \text{ union all } s$
- $\min(m, n)$ times in $r \text{ intersect all } s$
- $\max(0, m - n)$ times in $r \text{ except all } s$
Set operations: Duplicates

Union/Intersection/Except eliminate duplicates in the answer (the other SQL commands don’t) (e.g., try ‘select dept_name from instructor’).

Can use “union all” to retain duplicates.

NOTE: The duplicates are retained in a systematic fashion (for all SQL operations)

Suppose a tuple occurs $m$ times in $r$ and $n$ times in $s$, then, it occurs:

- $m + n$ times in $r$ union all $s$
- min($m,n$) times in $r$ intersect all $s$
- max(0, $m - n$) times in $r$ except all $s$
Today’s Plan

- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)

- Relational Algebra Continued
  - Different types of joins
  - Formal semantics of SQL
Relational Algebra

- Procedural language
- Six basic operators
  - select
  - project
  - union
  - set difference
  - Cartesian product
  - rename
- The operators take one or more relations as inputs and give a new relation as a result.
## Select Operation

Relation \( r \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ \sigma_{A=B \land D > 5} (r) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

**SQL Equivalent:**

```sql
select *
from r
where A = B and D > 5
```

*Unfortunate naming confusion*
Project

Relation r

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

\[ \Pi_{A,D} (r) \]

SQL Equivalent:

```sql
select distinct A, D
from r
```
**Set Union, Difference**

Relation $r, s$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td>α</td>
<td>2</td>
</tr>
<tr>
<td>β</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>2</td>
</tr>
<tr>
<td>β</td>
<td>3</td>
</tr>
</tbody>
</table>

$\alpha$ $\beta$ $1$ $2$ $1$ $3$

$r \cup s:$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td>α</td>
<td>2</td>
</tr>
<tr>
<td>β</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td>β</td>
<td>1</td>
</tr>
</tbody>
</table>

This is one case where duplicates are removed.

**SQL Equivalent:**

```
select * from r
union/except/intersect
select * from s;
```
# Cartesian Product

Relation `r`, `s`:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>β</td>
<td>20</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>β</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>β</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>γ</td>
<td>10</td>
<td>b</td>
</tr>
</tbody>
</table>

SQL Equivalent:

```sql
select distinct *
from r, s
```

Does not remove duplicates.
Allows us to name, and therefore to refer to, the results of relational-algebra expressions.

Allows us to refer to a relation by more than one name.

Example:

\[ \rho_X (E) \]

returns the expression \( E \) under the name \( X \).

If a relational-algebra expression \( E \) has arity \( n \), then

\[ \rho_X (A_1, A_2, ..., A_n) (E) \]

returns the result of expression \( E \) under the name \( X \), and with the attributes renamed to \( A_1, A_2, ..., A_n \).
Relational Algebra

- Those are the basic operations

- What about SQL Joins?
  - Compose multiple operators together
    \[ \sigma_{A=C}(r \times s) \]

- Additional Operations
  - Set intersection
  - Natural join
  - Division
  - Assignment
Additional Operators

- **Set intersection (\( \cap \))**
  - \( r \cap s = r - (r - s) \);
  - SQL Equivalent: intersect

- **Assignment (\( \leftarrow \))**
  - A convenient way to right complex RA expressions
  - Essentially for creating “temporary” relations
    - \( temp1 \leftarrow \prod_{R-S} (r) \)
  - SQL Equivalent: “create table as...”
Additional Operators: Joins

- **Natural join (⋈)**
  - A Cartesian product with equality condition on common attributes
  - Example:
    - if \( r \) has schema \( R(A, B, C, D) \), and if \( s \) has schema \( S(E, B, D) \)
    - Common attributes: \( B \) and \( D \)
    - Then:
      \[
      r \bowtie s = \prod_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \land r.D = s.D} (r \times s))
      \]

- **SQL Equivalent:**
  - select \( r.A, r.B, r.C, r.D, s.E \) from \( r, s \) where \( r.B = s.B \) and \( r.D = s.D \)
  - OR
  - select * from \( r \) natural join \( s \)
Additional Operators: Joins

- Equi-join
  - A join that only has equality conditions

- Theta-join ($\bowtie_\theta$)
  - $r \bowtie_\theta s = \sigma_\theta(r \times s)$

- Left outer join ($\bowland$)
  - Say $r(A, B), s(B, C)$
  - We need to somehow find the tuples in $r$ that have no match in $s$
  - Consider: $(r - \pi_{r.A, r.B}(r \bowland s))$
  - We are done:
    $$ (r \bowland s) \cup \rho_{\text{temp}}(A, B, C) \left( (r - \pi_{r.A, r.B}(r \bowland s)) \times \{\text{NULL}\} \right) $$
## Additional Operators: Join Variations

- **Tables:** $r(A, B)$, $s(B, C)$

<table>
<thead>
<tr>
<th>name</th>
<th>Symbol</th>
<th>SQL Equivalent</th>
<th>RA expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross product</td>
<td>$\times$</td>
<td>select * from $r$, $s$;</td>
<td>$r \times s$</td>
</tr>
<tr>
<td>natural join</td>
<td>$\bowtie$</td>
<td>natural join</td>
<td>$\pi_{r.A, r.B, s.C} \sigma_{r.B = s.B} (r \times s)$</td>
</tr>
<tr>
<td>theta join</td>
<td>$\bowtie_\theta$</td>
<td>from .. where $\theta$;</td>
<td>$\sigma_{\theta}(r \times s)$</td>
</tr>
<tr>
<td>equi-join</td>
<td>$\bowtie_\theta$</td>
<td><em>(theta must be equality)</em></td>
<td></td>
</tr>
<tr>
<td>left outer join</td>
<td>$r \Leftrightarrow s$</td>
<td>left outer join (with “on”)</td>
<td><em>(see previous slide)</em></td>
</tr>
<tr>
<td>full outer join</td>
<td>$r \Leftrightarrow s$</td>
<td>full outer join (with “on”)</td>
<td>$-$</td>
</tr>
<tr>
<td>(left) semijoin</td>
<td>$r \bowtie s$</td>
<td>none</td>
<td>$\pi_{r.A, r.B}(r \bowtie s)$</td>
</tr>
<tr>
<td>(left) antijoin</td>
<td>$r \nless s$</td>
<td>none</td>
<td>$r - \pi_{r.A, r.B}(r \bowtie s)$</td>
</tr>
</tbody>
</table>
Additional Operators: Division

- Suitable for queries that have “for all”
  - $r \div s$
- Think of it as “opposite of Cartesian product”
  - $r \div s = t \iff t \times s \subseteq r$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\beta$</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\beta$</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\alpha$</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\beta$</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\beta$</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
<td>$\gamma$</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

$\div$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>2</td>
</tr>
</tbody>
</table>

=  

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>$\beta$</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>$\beta$</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>10</td>
<td>b</td>
</tr>
</tbody>
</table>
Example Query

Find the largest salary in the university

Step 1: find instructor salaries that are less than some other instructor salary (i.e. not maximum)
   - using a copy of instructor under a new name $d$
     \[ \Pi_{\text{instructor}.\text{salary}} (\sigma \text{instructor}.\text{salary} < d, \text{salary} (\text{instructor} \rho_d (\text{instructor}))) \]

Step 2: Find the largest salary

\[ \Pi_{\text{salary}} (\text{instructor}) - \Pi_{\text{instructor}.\text{salary}} (\sigma \text{instructor}.\text{salary} < d, \text{salary} (\text{instructor} \rho_d (\text{instructor}))) \]
Example Queries

- Find the names of all instructors in the Physics department, along with the course_id of all courses they have taught

  - Query 1
    \[ \Pi_{instructor.ID, course_id} (\sigma_{dept_name=\text{“Physics”}} (\sigma_{instructor.ID=teaches.ID} (instructor \bowtie teaches))) ) \]

  - Query 2
    \[ \Pi_{instructor.ID, course_id} (\sigma_{instructor.ID=teaches.ID} (\sigma_{dept_name=\text{“Physics”}} (instructor) \bowtie teaches))) \]
By definition, *relations are sets*

- So → No duplicates allowed

Problem:

- Not practical to remove duplicates after every operation
- Why?

So...

- SQL by default does not remove duplicates

**SQL follows bag semantics, not set semantics**

- Implicitly we keep count of number of copies of each tuple
RA can only express \texttt{SELECT DISTINCT} queries

To express SQL, must extend RA to a \textit{bag} algebra

→ Bags (aka: \textit{multisets}) like sets, but can have duplicates

\textit{e.g.:} \{5, 3, 3\}

\textit{e.g.:} \textit{homes} =

<table>
<thead>
<tr>
<th>\texttt{cname}</th>
<th>\texttt{ccity}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson Smith</td>
<td>Brighton Perry</td>
</tr>
<tr>
<td>Johnson Smith</td>
<td>Brighton R.H.</td>
</tr>
</tbody>
</table>

Next: will define RA*: a \textit{bag} version of RA
1. $\sigma^*_p (r)$: *preserves copies in r*

\[\sigma^*_{\text{city} = \text{Brighton}} (\text{homes}) = \]

<table>
<thead>
<tr>
<th>cname</th>
<th>ccity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson</td>
<td>Brighton</td>
</tr>
<tr>
<td>Johnson</td>
<td>Brighton</td>
</tr>
</tbody>
</table>

2. $\pi^*_A (r)$: *no duplicate elimination*

\[\pi^*_{\text{cname}} (\text{homes}) = \]

<table>
<thead>
<tr>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson</td>
</tr>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Johnson</td>
</tr>
<tr>
<td>Smith</td>
</tr>
</tbody>
</table>
3. $r \cup^* s$: \textit{additive union}

\[
\begin{array}{c|c|c}
&A & B \\
1 & \alpha & \\
1 & \alpha & \\
2 & \beta & \\
\hline
r & & \\
\end{array}
\quad \cup^* 
\begin{array}{c|c|c}
&A & B \\
2 & \beta & \\
3 & \alpha & \\
1 & \alpha & \\
\hline
s & & \\
\end{array}
= 
\begin{array}{c|c|c}
&A & B \\
1 & \alpha & \\
1 & \alpha & \\
2 & \beta & \\
2 & \beta & \\
3 & \alpha & \\
\hline
\end{array}

4. $r -^* s$: \textit{bag difference}

\[e.g.: \quad r -^* s = \begin{array}{c|c|c}
&A & B \\
1 & \alpha & \\
\hline
\end{array} \quad s -^* r = \begin{array}{c|c|c}
&A & B \\
3 & \alpha & \\
\hline
\end{array}\]
5. \( r \times^* s \): cartesian product

\[
\begin{array}{ccc}
| A | B | C | \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\alpha</td>
<td>+</td>
</tr>
<tr>
<td>1</td>
<td>\alpha</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>\alpha</td>
<td>+</td>
</tr>
<tr>
<td>1</td>
<td>\alpha</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>\beta</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>\beta</td>
<td>-</td>
</tr>
</tbody>
</table>
\end{array}
\]
Formal Semantics of SQL

Query:
SELECT a_1, \ldots, a_n
FROM r_1, \ldots, r_m
WHERE p

Semantics:
\pi^*_{A_1, \ldots, A_n} (\sigma^*_p (r_1 \times * \ldots \times * r_m) ) \tag{1}

Query:
SELECT DISTINCT a_1, \ldots, a_n
FROM r_1, \ldots, r_m
WHERE p

Semantics:
What is the only operator to change in (1)?

\pi_{A_1, \ldots, A_n} (\sigma^*_p (r_1 \times * \ldots \times * r_m) ) \tag{2}
Set/Bag Operations Revisited

Set Operations
- UNION  ≡ U
- INTERSECT  ≡ ∩
- EXCEPT  ≡ -

Bag Operations
- UNION ALL  ≡ U*
- INTERSECT ALL  ≡ ∩*
- EXCEPT ALL  ≡ -*

Duplicate Counting:
Given m copies of t in r, n copies of t in s, how many copies of t in:

r UNION ALL s?
A: m + n

r INTERSECT ALL s?
A: min (m, n)

r EXCEPT ALL s?
A: max (0, m-n)