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

Basic Query Constructs

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);

Basic Query Constructs

Find the names of all instructors:

select *name*
from *instructor*

Renaming tables or output column names:

select *i.name, i.salary * 2* **as** *double_salary*
from *instructor i*
where *i.salary < 80000* **and** *i.name like '%g_';*

More complex expressions:

select *concat(name, concat(', ', dept_name))*
from *instructor;*

Careful with NULLs:

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

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

Multi-table Queries

Cartesian product:
select *
from *instructor, department*

Use predicates to only select "matching" pairs:
select *
from *instructor i, department d*
where *i.dept_name = d.dept_name;*

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:

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

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

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2009)
union
(**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2010);

In both:

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2009)
intersect
(**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2010);

In Fall 2009, but not in Spring 2010:

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2009)
except
(**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* **union all** *s*
- min(m,n) times in *r* **intersect all** *s*
- max(0, $m - n$) times in *r* **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

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

SQL Equivalent:

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

Unfortunate naming confusion

Project

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\Pi_{A,D}(r)$

A	D
α	7
α	7
β	3
β	10

A	D
α	7
β	3
β	10

SQL Equivalent:

select distinct A, D

from r

Set Union, Difference

Relation r, s

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$:

A	B
α	1
α	2
β	1
β	3

$r - s$:

A	B
α	1
β	1

Must be compatible schemas

What about intersection ?

Can be derived

$r \cap s = r - (r - s)$;

SQL Equivalent:

```
select * from r
union/except/intersect
select * from s;
```

This is one case where duplicates are removed.

Cartesian Product

Relation r, s

r:

A	B
α	1
β	2

s:

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

$r \times s$:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

SQL Equivalent:

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

Does not remove duplicates.

Rename Operation

- 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

$$\rho_{X(A1, A2, \ldots, An)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A2, \ldots, An$.

Relational Algebra

- Those are the basic operations

- What about SQL Joins ?
  - Compose multiple operators together

$$\sigma_{A=C}(r \; x \; s)$$

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

Additional Operators

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

- Assignment (←)
  - A convenient way to right complex RA expressions
  - Essentially for creating “temporary” relations
    - $temp1 \leftarrow \prod_{R\text{-}S}(r)$

  - SQL Equivalent: “create table as…”

Additional Operators: Joins

- Natural join ($\bowtie$)
  - 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:

$$r \bowtie s = \prod_{r.A,\ r.B,\ r.C,\ r.D,\ s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

- 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 (⟕)
  - Say *r(A, B), s(B, C)*
  - We need to somehow find the tuples in *r* that have no match in *s*
  - Consider: $(r - \pi_{r.A,\, r.B}(r \bowtie s))$
  - We are done:

$$(r \bowtie s) \quad \cup \quad \rho_{temp\ (A,\, B,\, C)}\left((r - \pi_{r.A,\, r.B}(r \bowtie s)) \times \{(NULL)\} \right)$$

Additional Operators: Join Variations

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

name	Symbol	SQL Equivalent	RA expression
cross product	$\times$	select * from r, s;	$r \times s$
natural join	$\bowtie$	natural join	$\pi_{r.A,\ r.B,\ s.C}\sigma_{r.B\ =\ s.B}(r\ x\ s)$
theta join	$\bowtie_\theta$	from .. where θ;	$\sigma_\theta(r\ x\ s)$
equi-join	$\bowtie_\theta$ *(theta must be equality)*		
left outer join	$r ⟕ s$	left outer join (with "on")	(see previous slide)
full outer join	$r ⟗ s$	full outer join (with "on")	–
(left) semijoin	$r \ltimes s$	none	$\pi_{r.A,\ r.B}(r \bowtie s)$
(left) antijoin	$r \triangleright s$	none	$r - \pi_{r.A,\ r.B}(r \bowtie s)$

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* $t \times s \subseteq r$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

÷

A	B
α	1
β	2

=

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

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*
      - $\Pi_{instructor.salary}(\sigma_{instructor.salary < d,salary}(instructor \times \rho_d(instructor)))$
  - Step 2: Find the largest salary
      - $\Pi_{salary}(instructor) - \Pi_{instructor.salary}(\sigma_{instructor.salary < d,salary}(instructor \times \rho_d(instructor)))$

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

    $$\Pi_{instructor.ID,course\_id}(\sigma_{dept\_name=\text{“Physics”}}(\sigma_{instructor.ID=teaches.ID}(instructor \times teaches)))$$

  - Query 2

    $$\Pi_{instructor.ID,course\_id}(\sigma_{instructor.ID=teaches.ID}(\sigma_{dept\_name=\text{“Physics”}}(instructor) \times teaches))$$

Duplicates

- 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

Formal Semantics of SQL

- RA can only express `SELECT DISTINCT` queries

- To express SQL, must extend RA to a bag algebra
  → *Bags (aka: multisets) like sets, but can have duplicates*

e.g: {5, 3, 3}

e.g: homes =

cname	ccity
Johnson	Brighton
Smith	Perry
Johnson	Brighton
Smith	R.H.

- Next: will define RA^*: a bag version of RA

Formal Semantics of SQL: RA*

1. σ^*_p (r): *preserves copies in* r

e.g: $\sigma^*_{city = Brighton}$ (homes) =

cname	ccity
Johnson	Brighton
Johnson	Brighton

2. $\pi^*_{A1, \ldots, An}$ (r): *no duplicate elimination*

e.g: π^*_{cname} (homes) =

cname
Johnson
Smith
Johnson
Smith

Formal Semantics of SQL: RA*

3. r ∪* s : *additive union*

A	B
1	α
1	α
2	β

r

∪*

A	B
2	β
3	α
1	α

s

=

A	B
1	α
1	α
2	β
2	β
3	α
1	α

4. r -* s: *bag difference*

e.g: r -* s =

A	B
1	α

s -* r =

A	B
3	α

Formal Semantics of SQL: RA*

5. r $\times^*$ s: *cartesian product*

A	B
1	α
1	α
2	β

$\times^*$

C
+
-

=

A	B	C
1	α	+
1	α	-
1	α	+
1	α	-
2	β	+
2	β	-

Formal Semantics of SQL

Query:

```
SELECT          a1, ….., an
FROM            r1, ….., rm
WHERE           p
```

Semantics: $\pi^*_{A1, \ldots, An} (\sigma^*_p (r_1 \times * \ldots \times * r_m))$ (1)

Query:

```
SELECT DISTINCT    a1, ….., an
FROM               r1, ….., rm
WHERE              p
```

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

$\pi_{A1, \ldots, An} (\sigma^*_p (r_1 \times * \ldots \times * r_m))$ (2)

Set/Bag Operations Revisited

- Set Operations
  - `UNION` ≡ $\cup$
  - `INTERSECT` ≡ $\cap$
  - `EXCEPT` ≡ -

Bag Operations

`UNION ALL` ≡ $\cup^*$
`INTERSECT ALL` ≡ $\cap^*$
`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:* $max\ (0, m\text{-}n)$