CMSC424: Database Design
Relational Model/SQL

Instructor: Amol Deshpande
amol@cs.umd.edu
Topics covered so far

- Why Databases
  - Data Modeling
  - Importance of abstraction/independence layers

- Relational Model
  - Relations, Tuples
  - Primary Keys, Foreign Keys
  - Referential Integrity Constraints

- Relational Algebra Operations

- SQL
  - Data Definition Language: How to create relations, change schemas, etc.
  - Data Manipulation Language: Simple single-table queries
Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
Basic Query Structure

\[
\text{select } A_1, A_2, \ldots, A_n \\
\text{from } r_1, r_2, \ldots, r_m \\
\text{where } P
\]

Attributes or expressions
Relations (or queries returning tables)
Predicates

Find the names of all instructors:
\[
\text{select name} \\
\text{from instructor}
\]

Apply some filters (predicates):
\[
\text{select name} \\
\text{from instructor} \\
\text{where salary > 80000 and dept_name = 'Finance'};
\]

Remove duplicates:
\[
\text{select distinct name} \\
\text{from instructor}
\]

Order the output:
\[
\text{select distinct name} \\
\text{from instructor} \\
\text{order by name asc}
\]
Basic Query Constructs

Find the names of all instructors:
```sql
select name
from instructor
```

Select all attributes:
```sql
select *
from instructor
```

Expressions in the select clause:
```sql
select name, salary < 100000
from instructor
```

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

A filter with a subquery:
```sql
select name
from instructor
where dept_name in (select dept_name from department where budget < 100000);
```
Basic Query Constructs

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_';
```

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)
Multi-table Queries

Cartesian product:
\[
\text{select * from instructor, department}
\]

Use predicates to only select “matching” pairs:
\[
\text{select * from instructor } i, \text{ department } d \\
\text{where } i.\text{dept}_\text{name} = d.\text{dept}_\text{name};
\]

Identical (in this case) to using a natural join:
\[
\text{select * from instructor natural join department;}
\]

Natural join does an equality on common attributes – doesn’t work here:
\[
\text{select * from instructor natural join advisor;}
\]

Instead can use “on” construct (or where clause as above):
\[
\text{select * from instructor join advisor on } (i._id = id);
\]
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:

\[
\text{(select course_id from section where semester = 'Fall' and year = 2009)}
\text{except}
\text{(select course_id from section where semester = 'Spring' and year = 2010)};
\]
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/Interception/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 $r \text{ except all } s$
Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
The “dirty little secret” of SQL
(major headache for query optimization)

Can be a value of any attribute

e.g: branch =

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bname</td>
<td>bcity</td>
<td>assets</td>
</tr>
<tr>
<td>Downtown</td>
<td>Boston</td>
<td>9M</td>
</tr>
<tr>
<td>Perry</td>
<td>Horseneck</td>
<td>1.7M</td>
</tr>
<tr>
<td>Mianus</td>
<td>Horseneck</td>
<td>.4M</td>
</tr>
<tr>
<td>Waltham</td>
<td>Boston</td>
<td>NULL</td>
</tr>
</tbody>
</table>

What does this mean?

*Unknown* We don’t know Waltham’s assets?
*Inapplicable* Waltham has a special kind of account without assets
*Withheld* We are not allowed to know
SQL: Nulls

Arithmetic Operations with Null

\[ n + \text{NULL} = \text{NULL} \]  
(similarly for all arithmetic ops: +, -, *, /, mod, ...)

e.g: branch =

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bname</td>
<td>bcity</td>
<td>assets</td>
</tr>
<tr>
<td>Downtown</td>
<td>Boston</td>
<td>9M</td>
</tr>
<tr>
<td>Perry</td>
<td>Horseneck</td>
<td>1.7M</td>
</tr>
<tr>
<td>Mianus</td>
<td>Horseneck</td>
<td>.4M</td>
</tr>
<tr>
<td>Waltham</td>
<td>Boston</td>
<td>NULL</td>
</tr>
</tbody>
</table>

SELECT bname, assets * 2 as a2
FROM branch

<table>
<thead>
<tr>
<th></th>
<th>a2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>18M</td>
</tr>
<tr>
<td>Perry</td>
<td>3.4M</td>
</tr>
<tr>
<td>Mianus</td>
<td>.8M</td>
</tr>
<tr>
<td>Waltham</td>
<td>NULL</td>
</tr>
</tbody>
</table>
**SQL: Nulls**

**Boolean Operations with Null**

\[ n < \text{NULL} = \text{UNKNOWN} \]

(similarly for all boolean ops: \( >, \leq, \geq, \neq, =, \ldots \))

e.g: branch =

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Boston</td>
<td>9M</td>
</tr>
<tr>
<td>Perry</td>
<td>Horseneck</td>
<td>1.7M</td>
</tr>
<tr>
<td>Mianus</td>
<td>Horseneck</td>
<td>.4M</td>
</tr>
<tr>
<td>Waltham</td>
<td>Boston</td>
<td>NULL</td>
</tr>
</tbody>
</table>

SELECT * FROM branch
WHERE assets = NULL

Counter-intuitive: \( \text{NULL} \times 0 = \text{NULL} \)

Counter-intuitive: select * from movies
where length \( \geq 120 \) or length \( \leq 120 \)
**SQL: Nulls**

**Boolean Operations with Null**

\[ n < \text{NULL} = \text{UNKNOWN} \quad \text{(similarly for all } boolean \text{ ops: } >, \ <=, \ >=, \ <>, \ =, \ ... \) \]

e.g.: \text{branch} =

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Boston</td>
<td>9M</td>
</tr>
<tr>
<td>Perry</td>
<td>Horseneck</td>
<td>1.7M</td>
</tr>
<tr>
<td>Mianus</td>
<td>Horseneck</td>
<td>.4M</td>
</tr>
<tr>
<td>Waltham</td>
<td>Boston</td>
<td>NULL</td>
</tr>
</tbody>
</table>

```sql
SELECT * =
FROM branch
WHERE assets IS NULL
```
**Boolean Operations with Unknown**

- \( n < \text{NULL} = \text{UNKNOWN} \)  
  (similarly for all *boolean ops*: \( >, \leq, \geq, <, =, \ldots \))

- \( \text{FALSE OR UNKNOWN} = \text{UNKNOWN} \)
- \( \text{TRUE AND UNKNOWN} = \text{UNKNOWN} \)

Intuition: substitute each of TRUE, FALSE for unknown. If different answer results, results is unknown

**Can write:**

- \( \text{SELECT ...} \)
- \( \text{FROM ...} \)
- \( \text{WHERE booleanexp IS UNKNOWN} \)

**UNKNOWN tuples are not included in final result**
Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
Aggregates

Other common aggregates:
max, min, sum, count, stdev, …

select count (distinct ID)
from teaches
where semester = ’Spring’ and year = 2010

Can specify aggregates in any query.
Find max salary over instructors teaching in S’10
select max(salary)
from teaches natural join instructor
where semester = ’Spring’ and year = 2010;

Aggregate result can be used as a scalar.
Find instructors with max salary:
select *
from instructor
where salary = (select max(salary) from instructor);

Find the average salary of instructors in the Computer Science
select avg(salary)
from instructor
where dept_name = ‘Comp. Sci’;

Other common aggregates:
max, min, sum, count, stdev, …

Aggregate result can be used as a scalar. Find instructors with max salary:

```
select *
from instructor
where salary = (select max(salary) from instructor);
```

Following doesn’t work:

```
select *
from instructor
where salary = max(salary);
```

```
select name, max(salary)
from instructor
where salary = max(salary);
```
Split the tuples into groups, and compute the aggregate for each group

```
select dept_name, avg(salary)
from instructor
group by dept_name;
```
Attributes in the select clause must be aggregates, or must appear in the group by clause. Following wouldn’t work:

```sql
select dept_name, ID, avg(salary)
from instructor
group by dept_name;
```

“having” can be used to select only some of the groups:

```sql
select dept_name, avg(salary)
from instructor
group by dept_name
having avg(salary) > 42000;
```
Given

\[
\text{branch} =
\begin{array}{|c|c|c|}
\hline
\text{bname} & \text{bcity} & \text{assets} \\
\hline
\text{Downtown} & \text{Boston} & 9M \\
\text{Perry} & \text{Horseneck} & 1.7M \\
\text{Mianus} & \text{Horseneck} & 0.4M \\
\text{Waltham} & \text{Boston} & \text{NULL} \\
\hline
\end{array}
\]

Aggregate Operations

\[
\text{SELECT SUM (assets) =}
\]
\[
\begin{array}{|c|}
\hline
\text{SUM} \\
\hline
11.1 M \\
\hline
\end{array}
\]

\text{NULL is ignored for SUM}

\text{Same for AVG (3.7M), MIN (0.4M), MAX (9M)}

\text{Also for COUNT(assets) -- returns 3}

\text{But COUNT (*) returns}

\[
\begin{array}{|c|}
\hline
\text{COUNT} \\
\hline
4 \\
\hline
\end{array}
\]
Given

branch =

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
</tr>
</thead>
</table>

SELECT SUM (assets) =  
FROM branch

• *Same as AVG, MIN, MAX*
• *But COUNT (assets) returns*
Summary

- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics

- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)