Overview of modeling

SQL (Chapter 3)
- Basic Data Definition (3.2)
- Basic Queries (3.3-3.5)
- Null values (3.6)
- Aggregates (3.7)

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

Basic Query Structure

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

Attributes or expressions
Relations (or queries returning tables)
Predicates

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

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

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

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

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

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

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

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

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

Wouldn’t return the instructor with NULL salary (if any)
Multi-table Queries

Use predicates to only select “matching” pairs:

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

Cartesian product:

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

Almost same (in this case) to using natural join:

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

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

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

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

```sql
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.id = teaches.course_id;
```

Beware of unintended common names (happens often)
You may think the following query has

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

I prefer avoiding “natural joins” for that reason
Set operations

Find courses that ran in Fall 2009 or Spring 2010

\[
\begin{align*}
&\text{(select course_id from section where semester = ‘Fall’ and year = 2009)} \\
&\text{union} \\
&\text{(select course_id from section where semester = ‘Spring’ and year = 2010});
\end{align*}
\]

In both:

\[
\begin{align*}
&\text{(select course_id from section where semester = ‘Fall’ and year = 2009)} \\
&\text{intersect} \\
&\text{(select course_id from section where semester = ‘Spring’ and year = 2010});
\end{align*}
\]

In Fall 2009, but not in Spring 2010:

\[
\begin{align*}
&\text{(select course_id from section where semester = ‘Fall’ and year = 2009)} \\
&\text{except} \\
&\text{(select course_id from section where semester = ‘Spring’ and year = 2010);}
\end{align*}
\]

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 \( r \text{ except all } s \)
Outline

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

SQL: Nulls

The “dirty little secret” of SQL  
(major headache for query optimization)

Can be a value of any attribute

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>

What does this mean?

(\textit{not known}) \quad \text{We don't know Waltham's assets}

(\textit{inapplicable}) \quad \text{Waltham has a special kind of account without assets}

(\textit{withheld}) \quad \text{We are not allowed to know}
SQL: Nulls

Arithmetic Operations with NULL

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

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 bname, assets * 2 as a2
FROM branch

= | bname    | a2   |
<table>
<thead>
<tr>
<th></th>
<th></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

Arithmetic Operations with NULL

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

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 IS NULL

= | bname    | bcity     | assets |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Waltham</td>
<td>Boston</td>
<td>NULL</td>
</tr>
</tbody>
</table>
SQL: Nulls

Counter-intuitive: NULL * 0 = NULL

Counter-intuitive: select * from movies
where length >= 120 or length <= 120

SQL: Unknown

Boolean Operations with Unknown

n < NULL = UNKNOWN  
(similarly for all boolean ops: >, <=, >=, <>, =, ...)

FALSE OR UNKNOWN = UNKNOWN
TRUE AND UNKNOWN = UNKNOWN

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

UNKNOWN OR UNKNOWN = UNKNOWN
UNKNOWN AND UNKNOWN = UNKNOWN
NOT (UNKNOWN) = UNKNOWN

Can write:

SELECT ...
FROM ...
WHERE booleanexp IS UNKNOWN
Overview of modeling

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

SQL (Chapter 3)
- Basic Data Definition (3.2)
- Basic Queries (3.3-3.5)
- Null values (3.6)
- Aggregates (3.7)

Aggregates

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

Finding the average salary of instructors in the Computer Science:

```
select avg(salary)
from instructor
where dept_name = 'Comp. Sci';
```

Finding instructors with max salary:

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

In a join:

```
select max(salary)
from teaches natural join instructor
where semester = 'Spring' and year = 2010;
```
Aggregates

Aggregate result can be used as a scalar.

Find instructors with max salary:

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

The following do not work:

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

```sql
select name, max(salary)
from instructor;
```

Split the tuples into groups, and compute the aggregate for each group

```sql
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

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

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

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

*having* used to select from aggregated rows

*where* used to select non-aggregated rows

---

### Aggregates and NULLs

Given

```
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>

Aggregate Operations

```
SELECT SUM(assets) =
FROM branch
```

<table>
<thead>
<tr>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1M</td>
</tr>
</tbody>
</table>

*NULL is ignored for SUM*

*Same for AVG (3.7M), MIN (0.4M), MAX (9M)*

Also for COUNT(assets) -- returns 3

<table>
<thead>
<tr>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
Aggregates and NULLs

Given

\[
\text{branch} = \begin{array}{ccc}
\text{bname} & \text{bcity} & \text{assets} \\
\hline
\end{array}
\]

\[
\text{SELECT SUM (assets) = SUM NULL FROM branch}
\]

- *Same as AVG, MIN, MAX*
- *But COUNT (assets) returns 0*

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)
  - Advanced operators
With Clause

- The **with** clause provides a way of defining a temporary table (or “view”) whose definition is available only to the query in which the **with** clause occurs.
- Find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
     from department)
select *
from department, max_budget
where department.budget = max_budget.value;
```

With Clause, cont

- **WITH**
  - b AS ((SELECT * FROM borders) UNION (SELECT country2,country1...)
  - cd AS (SELECT code FROM country WHERE name='Germany'),
  - b1 AS (SELECT DISTINCT b.country1 FROM b,cd WHERE b.country2 = cd.code),
  - b2 AS (SELECT DISTINCT b.country1 FROM b,b1 WHERE (b.country2 = b1.country1)),
  - b3 AS ((select * from b2) minus (select * from b1))
  - SELECT name FROM b3,country WHERE country.code = b3.country1;