Outline

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

Basic Query Structure

```
sel ect \ A_1, A_2, \ldots, A_n \from \ r_1, r_2, \ldots, r_m \where P
```

- Select columns or expressions
- From tables (or queries returning tables)
- Where predicates

Examples:
- Find the names of all instructors:
  ```
  select name 
  from instructor 
  ```
- Apply some filters (predicates):
  ```
  select name 
  from instructor 
  where salary > 55000 and dept_name = 'Statistics';
  ```
- Remove duplicates:
  ```
  select distinct name 
  from instructor 
  ```
- Order the output:
  ```
  select distinct name 
  from instructor 
  order by name desc 
  ```
Basic Query Constructs

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

Expressions in the select clause:
\[ \text{select name, salary < 55000 from instructor} \]

A filter with a subquery:
\[ \text{select name from instructor where dept_name in (select dept_name from department where budget < 800000);} \]

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

Renaming tables or output column names:
\[ \text{select i.name, i.salary * 2 as double_salary from instructor i where i.salary < 55000 and i.name like ‘%g_’;} \]

More complex expressions:
\[ \text{select concat(name, ‘’, dept_name)) from instructor;} \]

\[ \text{select name from instructor where salary < 100000 or salary >= 100000;} \]

\[ \text{Wouldn’t return the instructor with NULL salary (if any)} \]
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
```

Above almost same here 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 the same result:

```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 Spring 2002 or Spring 2010:

\[
\text{(select course_id from section where semester = 'Spring' and year = 2002)}
\]
union
\[
\text{(select course_id from section where semester = 'Spring' and year = 2010)}
\]

In both:
\[
\text{(select course_id from section where semester = 'Spring' and year = 2002)}
\]
intersect
\[
\text{(select course_id from section where semester = 'Spring' and year = 2010)}
\]

In Fall 2009, but not in Spring 2010:
\[
\text{(select course_id from section where semester = 'Spring' and year = 2002)}
\]
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 \( r \, \text{union all} \, s \)
- \( \min(m,n) \) times in \( r \, \text{intersect all} \, s \)
- \( \max(0, m - n) \) times in \( r \, \text{except all} \, s \)
SQL: Nulls

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

Can be a value of any attribute

\[ \text{e.g: } \text{branch} = \]

<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>

What does this mean?

(not known) 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

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

\[ \text{e.g: branch} = \]

\[
\begin{array}{|c|c|c|}
\hline
\text{bname} & \text{bcity} & \text{assets} \\
\hline
\text{Downtown} & \text{Boston} & 9M \\
\text{Perry} & \text{Horseneck} & 1.7M \\
\text{Mianus} & \text{Horseneck} & .4M \\
\text{Waltham} & \text{Boston} & \text{NULL} \\
\hline
\end{array}
\]

\[ \text{SELECT bname, assets * 2 as a2 = } \]

\[ \text{FROM branch} \]

\[
\begin{array}{|c|c|}
\hline
\text{bname} & \text{a2} \\
\hline
\text{Downtown} & 18M \\
\text{Perry} & 3.4M \\
\text{Mianus} & .8M \\
\text{Waltham} & \text{NULL} \\
\hline
\end{array}
\]
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

SQL: Nulls

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

Counter-intuitive: select * from movies where length \( \geq 120 \) or length \( \leq 120 \)
**Boolean Operations with Unknown**

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

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

  Intuition: substitute each of TRUE, FALSE for unknown. If different answers result, result is unknown

- \( \text{UNKNOWN OR UNKNOWN} = \text{UNKNOWN} \)
- \( \text{UNKNOWN AND UNKNOWN} = \text{UNKNOWN} \)
- \( \text{NOT (UNKNOWN)} = \text{UNKNOWN} \)

*Can write:*

```
SELECT ...  
FROM ...  
WHERE booleanexp IS UNKNOWN
```

---

**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)
Aggregates

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

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

The following do not work:

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

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;
```