Relational Data Processing

Everything Data
CompSci 290.01 Spring 2014
Announcements (Thu. Jan. 23)

• Two UTAs on board!
  – Dan Deng
  – Kathleen Miyoko Oshima

• **Homework #3** will be posted by tonight
  – Due midnight Monday

• We are working on a cloud-based solution for those with slow VMs—stay tuned
Structure is good

• More structure ➞ easier, more powerful analysis
• What’s your favorite structure?

Allows sorting, filtering, grouping, counting, summing, …
But remember this exercise?

Find the longest-serving current members of the Congress

• A member may serve multiple terms
• So data has a nested structure
Spreadsheet chokes…

… on this and other more complex structures
How do we structure data now?

One table ➔ multiple tables
Persons & their roles

<table>
<thead>
<tr>
<th>persons</th>
<th>person_roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>One row per person</td>
<td>One row for each term served by a person</td>
</tr>
</tbody>
</table>

- Birthday
- Gender
- Start/end of a term
- Party
- State
- House/senate?
- Person ID
- Name
Relational data model

How is data structured/constrained?

• Organize data in tables (AKA relations)
  – Each table has a list of (typed) columns
    • Data is stored as rows
    • Each row has a value for every column

• Declare structure + constraints as schema

How is data queried/updated?

• A “declarative” language called SQL
  – Say what result you want, not how to compute it
CREATE TABLE persons (  
id CHAR(10) NOT NULL PRIMARY KEY,  
id_govtrack INTEGER NOT NULL UNIQUE,  
id_lis CHAR(4) UNIQUE,  
first_name VARCHAR(50) NOT NULL,  
middle_name VARCHAR(50),  
last_name VARCHAR(50) NOT NULL,  
birthday DATE,  
gender CHAR(1)  
CHECK (gender IS NULL OR gender IN ('F', 'M'))  
);

Key constraint: no two rows in this table can have the same key value

The database will use the PRIMARY key to identify rows

String of length 10

Cannot be NULL, a special value used to indicate missing or inapplicable values

Specifies additional constraints on the column value
person_roles schema

CREATE TABLE person_roles (  
  person_id CHAR(10) NOT NULL REFERENCES persons(id),  
  type CHAR(3) NOT NULL CHECK (type IN ('rep', 'sen')),  
  start_date DATE NOT NULL,  
  end_date DATE NOT NULL,  
  state CHAR(2) NOT NULL REFERENCES states(id),  
  district INTEGER  
    CHECK ((type = 'rep' AND district IS NOT NULL) OR  
        (type = 'sen' AND district IS NULL)),  
  party VARCHAR(20)  
);
# Example data

## persons

<table>
<thead>
<tr>
<th>id</th>
<th>id_govtrack</th>
<th>id_lis</th>
<th>first_name</th>
<th>middle_name</th>
<th>last_name</th>
<th>birthday</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000369</td>
<td>412500</td>
<td>412533</td>
<td>Mark</td>
<td>E.</td>
<td>Amodei</td>
<td>1958-06-12</td>
<td>M</td>
</tr>
<tr>
<td>D000622</td>
<td>412533</td>
<td></td>
<td>Tammy</td>
<td></td>
<td>Duckworth</td>
<td>1968-03-12</td>
<td>F</td>
</tr>
</tbody>
</table>

## person_roles

<table>
<thead>
<tr>
<th>person_id</th>
<th>type</th>
<th>start_date</th>
<th>end_date</th>
<th>state</th>
<th>district</th>
<th>party</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000369</td>
<td>rep</td>
<td>2013-01-03</td>
<td>2015-01-03</td>
<td>NV</td>
<td>2</td>
<td>Republican</td>
</tr>
<tr>
<td>A000369</td>
<td>rep</td>
<td>2011-09-13</td>
<td>2013-01-03</td>
<td>NV</td>
<td>2</td>
<td>Republican</td>
</tr>
<tr>
<td>D000622</td>
<td>rep</td>
<td>2013-01-03</td>
<td>2015-01-03</td>
<td>IL</td>
<td>8</td>
<td>Democrat</td>
</tr>
</tbody>
</table>

...
For simplicity…

Assume we have this “table”:

– Columns in the primary key are underlined

```sql
cur_members(id, first_name, last_name,
            gender, birthday,
            type, party, state)
```

We will come back to how to create this “table” later
Show me a table... sorted

• List all current members of the Congress

```
SELECT *  
FROM cur_members;  
```

A shorthand for “all columns”

Marks the end of the query

A table

• **Sorting** options

  - ... **ORDER BY** birthday;  (Default is ascending order)
  - ... ORDER BY birthday DESC;
  - ... ORDER BY type, party;

cur_members(id, first_name, last_name, gender, birthday, type, party, state)
Picking columns

• AKA *projection*

```sql
SELECT id, first_name, last_name, state, type,
    (date_part('year', current_date) - date_part('year', birthday)) AS age
FROM cur_members
ORDER BY age;
```

You can compute a new column to output ... and give it a name

cur_members(id, first_name, last_name, gender, birthday, type, party, state)
Picking rows

- AKA *filtering* or *selection*

```sql
SELECT *
FROM cur_members
WHERE type = 'sen' AND birthday >= '1950-01-01';
```

Strings are enclosed by single quotes

Comparison, not assignment

... AND ...
... OR ...
NOT (...)

cur_members(id, first_name, last_name, gender, birthday, type, party, state)
**Grouping and aggregating rows**

```sql
SELECT party, COUNT(*)
FROM cur_members
GROUP BY party;
```

- Put members of the same party in one group
- Count the size of each group
More *grouping/aggregation*

```sql
SELECT party, gender, COUNT(*),
    AVG(date_part('year', current_date) - date_part('year', birthday))
FROM cur_members
GROUP BY party, gender;
```

Rows now must match on *both* columns to be in the same group

Again, one output row per group

Other aggregation functions include `SUM`, `MAX`, `MIN`

cur_members(id, first_name, last_name, gender, birthday, type, party, state)
Joining tables

• How did we get `cur_members(id, first_name, last_name, gender, birthday, type, party, state)`?

• Need to “join” tables together

```sql
persons(id, id_govtrack, id_lis, first_name, last_name, birthday, gender)
person_roles(person_id, type, start_date, end_date, state, district, party)
```
A SQL Query walks into a bar. In the corner of the bar are two tables. The Query walks up to the tables and asks,

“Mind if I join you?”

Join = pairing “related” rows

**persons**

<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000369</td>
<td>Mark</td>
<td>Amodei</td>
<td>...</td>
</tr>
<tr>
<td>D000622</td>
<td>Tammy</td>
<td>Duckworth</td>
<td>...</td>
</tr>
</tbody>
</table>

**person Roles**

<table>
<thead>
<tr>
<th>person_id</th>
<th>...</th>
<th>start_date</th>
<th>end_date</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000369</td>
<td></td>
<td>2013-01-03</td>
<td>2015-01-03</td>
<td>...</td>
</tr>
<tr>
<td>A000369</td>
<td></td>
<td>2011-09-13</td>
<td>2013-01-03</td>
<td>...</td>
</tr>
<tr>
<td>D000622</td>
<td></td>
<td>2013-01-03</td>
<td>2015-01-03</td>
<td>...</td>
</tr>
</tbody>
</table>

“Join condition” is persons.id = person_roles.person_id

Output table:

<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>...</th>
<th>person_id</th>
<th>...</th>
<th>start_date</th>
<th>end_date</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000369</td>
<td>Mark</td>
<td>Amodei</td>
<td>...</td>
<td>A000369</td>
<td></td>
<td>2013-01-03</td>
<td>2015-01-03</td>
<td>...</td>
</tr>
<tr>
<td>A000369</td>
<td>Mark</td>
<td>Amodei</td>
<td>...</td>
<td>A000369</td>
<td></td>
<td>2011-09-13</td>
<td>2013-01-03</td>
<td>...</td>
</tr>
<tr>
<td>D000622</td>
<td>Tammy</td>
<td>Duckworth</td>
<td>...</td>
<td>D000622</td>
<td></td>
<td>2013-01-03</td>
<td>2015-01-03</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
cur_members in SQL

```
CREATE VIEW cur_members AS
SELECT p.id, p.first_name, p.last_name, p.gender, p.birthday,
     r.type, r.party, r.state
FROM persons p, person_roles r
WHERE p.id = r.person_id
AND r.end_date >= '2014-01-01';
```

Want to “save” your output for later querying?
Create a view—a “virtual” table

List tables to be joined
Join condition
Selection condition local to r

Make p an alias for persons—think of it as a variable iterating through persons rows
Putting it together

**SELECT** columns or expressions

(or for each group of them if query has grouping/aggregation)

**FROM** tables

1. Generate all combinations of rows, one from each table; each combination forms a “wide row”

2. Filter—keep only “wide rows” satisfying conditions

**WHERE** conditions

3. Group—“wide rows” with matching values for columns go into the same group

**GROUP BY** columns

4. Compute one output row for each “wide row” (or for each group of them if query has grouping/aggregation)

**ORDER BY** output columns;

5. Sort the output rows
Subqueries and **LIMIT**

- Find the ten longest serving members

```sql
SELECT id, first_name, last_name, birthday, (SELECT SUM(end_date - start_date) 
FROM person_roles r 
WHERE r.person_id = p.id) AS duration 
FROM persons p 
ORDER BY duration DESC LIMIT 10;
```

Pretend that for every `p` in `persons` we examine, we run the subquery with `p`’s id value

Just give me the first 10 rows
One more example

How does he vote?
Say, comparing with

- Pelosi (D, CA), minority leader
- Cantor (R, VA), majority leader

Rep. (& Prof.) David Price (D, NC 4th District)

Image credits:
## Expanded schema

<table>
<thead>
<tr>
<th>persons</th>
<th>person_roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>person_id</td>
</tr>
<tr>
<td>...</td>
<td>type</td>
</tr>
<tr>
<td>first_name</td>
<td>start_date</td>
</tr>
<tr>
<td>last_name</td>
<td>end_date</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>votes</th>
<th>person_voted</th>
</tr>
</thead>
<tbody>
<tr>
<td>vote_id</td>
<td>vote_id</td>
</tr>
<tr>
<td>category</td>
<td>person_id</td>
</tr>
<tr>
<td>chamber</td>
<td>vote</td>
</tr>
<tr>
<td>session</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Here we go…

WITH creates a temporary view for the query that follows

WITH votes_compare(vote_id, vote1, vote2) AS
(SELECT v1.vote_id, v1.vote, v2.vote
FROM votes v, persons p1, persons p2, person_votes v1, person_votes v2
WHERE v.chamber = 'h' AND v.session = 2013
    AND p1.last_name = 'Price' AND p2.last_name = 'Pelosi'
    AND v1.person_id = p1.id AND v2.person_id = p2.id
    AND v1.vote_id = v2.vote_id AND v.id = v1.vote_id)

SELECT COUNT(*) AS agree,
    (SELECT COUNT(*) FROM votes_compare) AS total,
    COUNT(*)*100.00 / (SELECT COUNT(*) FROM votes_compare) AS percent
FROM votes_compare
WHERE vote1 = vote2;
... and the results are:

| agree | total | percent          |
|-------+-------+------------------|
| 717   | 1282  | 55.9282371294851794 |

vs. Pelosi (D, CA), minority leader

What's going on?
Isn't Price a Democrat?

| agree | total | percent          |
|-------+-------+------------------|
| 768   | 1282  | 59.9063962558502340 |

vs. Cantor (R, VA), majority leader

It's your job to clear Prof. Price's name!

Dear chain-mailer: This is an exercise teaching students how to catch subtle errors in SQL queries—the numbers here are WRONG and just don't grab them mindlessly