SQL Queries

CPSC 315 – Programming Studio
Fall 2011
Project 1, Lecture 4

Slides adapted from those used by Jeffrey Ullman, via Jennifer Welch

Modifying the Database

- Data Manipulation Language
- Given a schema, must “populate” the database with actual data
- Insert, Delete, Modify

Insertion

- INSERT command:
  INSERT INTO <Relation>
  VALUES (<value list>);
- Can specify only certain attributes in Relation
  Relation(<attribute list>)
- Instead of values, can have subquery

Insertion Example

- Senator(Name,Party,State,Years)
  INSERT INTO Senator
  VALUES (Jill Smith, Republican, NY, 5);
  INSERT INTO Senator(Name, State)
  VALUES (Jill Smith, NY);
Deletion

- Delete from relation according to condition
  
  `DELETE FROM <Relation> WHERE <condition>;`

- Example: delete Texas Senators:
  
  `DELETE FROM Senator WHERE State = 'TX';`

Modification

- Update subset according to condition
  
  `UPDATE <Relation> SET <list of attribute assignments> WHERE <condition>;;`

- Example: Joe Lieberman becomes Independent
  
  `UPDATE Senator SET Party = 'Independent' WHERE Name = 'Joseph Lieberman';`

Queries

- The heart of SQL

- Queries can form portion of other commands
  - e.g. INSERT results of a query into a table

- Form:
  - SELECT attributes
  - FROM relation(s)
  - WHERE condition

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Joe Adams</td>
<td>Democrat</td>
<td>NJ</td>
<td>0</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

Senator:

```
SELECT Name 
FROM Senator 
WHERE Party = 'Republican';
```

Result:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
</tr>
<tr>
<td>Jim Brown</td>
</tr>
</tbody>
</table>
Statement Processing

- Begin with the relation(s) in the FROM clause
  - Can be the result of another query!
- Apply selection condition in WHERE clause
  - Can potentially be very complex, and include subqueries
- Get the attributes given in (more generally, apply a projection to) the SELECT clause
- Process: iterate through all tuples in FROM, checking vs. WHERE, and for those that match, apply the SELECT

SELECT Clause - *

- Can use a * for SELECT to indicate all attributes given in the relation listed in FROM.

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Joe Adams</td>
<td>Democrat</td>
<td>NJ</td>
<td>0</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

SELECT Clause - AS

- Can use AS to rename attributes in result

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Joe Adams</td>
<td>Democrat</td>
<td>NJ</td>
<td>0</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

SELECT Clause - Expression

- Can include expressions in SELECT Clause

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Joe Adams</td>
<td>Democrat</td>
<td>NJ</td>
<td>0</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>DaysInOffice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>1825</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>5475</td>
</tr>
</tbody>
</table>
SELECT Clause - Constants

- Can include constant attributes
- Senator:

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Joe Adams</td>
<td>Democrat</td>
<td>NJ</td>
<td>0</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

Query:
SELECT Name, 'Senator' AS OfficeHeld
FROM Senator
WHERE Party = 'Republican';

Result:
<table>
<thead>
<tr>
<th>Name</th>
<th>OfficeHeld</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Senator</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Senator</td>
</tr>
</tbody>
</table>

Aggregations

- SUM, AVG, COUNT, MIN, MAX
  - COUNT(*) counts number of tuples
- Applied to column in SELECT clause
- Use DISTINCT to eliminate duplicates
- NULLs are ignored
- If Aggregation is used, every selected column must be aggregated or in the GROUP BY list

Grouping Aggregations

- Adding GROUP BY <attribute> at the end will apply aggregation only to group
  - e.g. to get the total number of U.S. Representatives from each state:

```
SELECT State, COUNT(*)
FROM USRepresentatives
GROUP BY State
```

HAVING

- Can restrict GROUP using HAVING
  - HAVING can refer to the FROM clause and its attributes
  - e.g. Count representatives by state, only if all representatives have 3 years experience

```sql
SELECT State, COUNT(*)
FROM USRepresentatives
GROUP BY State
   HAVING MIN(Years) > 3
```
WHERE Clause – Complex Expressions

- Can include NOT, AND, OR operators
- Senator:

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Joe Adams</td>
<td>Democrat</td>
<td>NJ</td>
<td>0</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

- Query: SELECT * FROM Senator WHERE Party = 'Republican' OR Years > 3;

- Result:

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

WHERE Clause – other effects

- Order of operations, including parentheses
- LIKE: String comparisons with wildcards
  - % means any string
  - _ means any character

WHERE Clause – NULL values

- Tuples may contain NULL values
  - Undefined/Unknown
  - Inapplicable
- All conditions evaluate to either TRUE, FALSE, or UNKNOWN
- Comparisons to NULL are UNKNOWN
- Tuples selected only if TRUE

3-valued Logic

- Can think of values as
  - TRUE = 1
  - FALSE = 0
  - UNKNOWN = ½
- Operations would be
  - OR = MAX
  - AND = MIN
  - NOT = 1-x
- Example: \((T \land ((\neg U \lor F) \land \neg (U \lor T)))\)
3-valued Logic

- Can think of values as
  - TRUE = 1
  - FALSE = 0
  - UNKNOWN = ½

- Operations would be
  - OR = MAX
  - AND = MIN
  - NOT = 1-x

- Example: \((T \land ((\neg U \lor F) \land \neg (U \lor T)))\)
  
  \[
  \max(1- \frac{1}{2}, 0) = \max(\frac{1}{2}, 0) = \frac{1}{2} = U
  \]
3-valued Logic

- Can think of values as
  - TRUE = 1
  - FALSE = 0
  - UNKNOWN = \( \frac{1}{2} \)
- Operations would be
  - OR = MAX
  - AND = MIN
  - NOT = 1-x
- Example: \((T \land (U \land \neg T))\)
  \[ \text{MIN}(\frac{1}{2}, 1-1) = \text{MIN}(\frac{1}{2},0) = 0 = F \]

3-valued Logic

- Can think of values as
  - TRUE = 1
  - FALSE = 0
  - UNKNOWN = \( \frac{1}{2} \)
- Operations would be
  - OR = MAX
  - AND = MIN
  - NOT = 1-x
- Example: \((T \land F)\)
  \[ \text{MIN}(0,1) = 0 = F \]

3-valued Logic

- Can think of values as
  - TRUE = 1
  - FALSE = 0
  - UNKNOWN = \( \frac{1}{2} \)
- Operations would be
  - OR = MAX
  - AND = MIN
  - NOT = 1-x
- Example: \((T \land F)\)
  \[ \text{MIN}(0,1) = 0 = F \]

3-valued Logic

- Can think of values as
  - TRUE = 1
  - FALSE = 0
  - UNKNOWN = \( \frac{1}{2} \)
- Operations would be
  - OR = MAX
  - AND = MIN
  - NOT = 1-x
- Example: \((T \land ((\neg U \lor F) \land \neg (U \lor T)))\)
**Unexpected Results for NULLs**

- WHERE (Years > 2) OR (Years < 3)
- This should “cover” all cases
- If Years is NULL
  - Years > 2 is UNKNOWN
  - Years < 3 is UNKNOWN
  - So the OR is UNKNOWN
  - And thus the tuple is NOT selected!

---

**WHERE Clause – IN operator**

- <tuple> IN <relation>
  - TRUE iff the tuple is a member of the relation

```
SELECT *
FROM ElectedOfficial
WHERE Name IN USRep
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chet Edwards</td>
<td>Democrat</td>
</tr>
<tr>
<td>John Cornyn</td>
<td>Republican</td>
</tr>
<tr>
<td>John Adams</td>
<td>Federalist</td>
</tr>
<tr>
<td>Ron Paul</td>
<td>Republican</td>
</tr>
</tbody>
</table>

**Result**

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chet Edwards</td>
<td>Democrat</td>
</tr>
<tr>
<td>Ron Paul</td>
<td>Republican</td>
</tr>
</tbody>
</table>

---

**WHERE Clause – EXISTS operator**

- EXISTS (<relation>)
  - TRUE iff the relation is not empty relation

```
SELECT *
FROM ElectedOfficial
WHERE EXISTS(USRep)
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chet Edwards</td>
<td>Democrat</td>
</tr>
<tr>
<td>John Cornyn</td>
<td>Republican</td>
</tr>
<tr>
<td>John Adams</td>
<td>Federalist</td>
</tr>
<tr>
<td>Ron Paul</td>
<td>Republican</td>
</tr>
</tbody>
</table>

**Result**

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chet Edwards</td>
<td>Democrat</td>
</tr>
<tr>
<td>John Cornyn</td>
<td>Republican</td>
</tr>
<tr>
<td>John Adams</td>
<td>Federalist</td>
</tr>
<tr>
<td>Ron Paul</td>
<td>Republican</td>
</tr>
</tbody>
</table>

---

**EXISTS (and other) operators**

- Usually applied to the results of a subquery
- Example: is any Senator a Whig?

```
EXISTS(
    SELECT *
    FROM Senator
    WHERE Party = 'Whig'
)
```
WHERE Clause – ANY and ALL operators

- \( x = \text{ANY}(\text{<relation>}) \)
  - TRUE iff \( x \) is equal to at least one tuple in the relation
- \( x = \text{ALL}(\text{<relation>}) \)
  - TRUE iff \( x \) is equal to all tuples in the relation
- The = can also be >, >=, <, <=, <>
- The relation should have only one attribute

Example: ANY

```
SELECT *
FROM ElectedOfficial
WHERE Party = \text{ANY} (\text{CurrentParties})
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chet Edwards</td>
<td>Democrat</td>
</tr>
<tr>
<td>John Cornyn</td>
<td>Republican</td>
</tr>
<tr>
<td>John Adams</td>
<td>Federalist</td>
</tr>
<tr>
<td>Ron Paul</td>
<td>Republican</td>
</tr>
</tbody>
</table>

Example: ALL

```
SELECT *
FROM Senator
WHERE Years > \text{ALL} (\text{YearsPresidentsInSenate})
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>State</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill Smith</td>
<td>Republican</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Joe Adams</td>
<td>Democrat</td>
<td>NJ</td>
<td>0</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Democrat</td>
<td>CT</td>
<td>9</td>
</tr>
<tr>
<td>Jim Brown</td>
<td>Republican</td>
<td>PA</td>
<td>15</td>
</tr>
</tbody>
</table>

UNION, INTERSECT, DIFFERENCE

- Can combine subqueries with Boolean operations
  - e.g. (subquery) UNION (subquery)
- Default: duplicates are removed by these operations unless ALL is included
  - (subquery) INTERSECT ALL (subquery)
- Likewise, can remove duplicates in normal SELECT by including DISTINCT
  - SELECT DISTINCT Years ...
"Bag" vs. "Set" semantics

- Items are in a "bag"
  - Duplicates OK
- Items are in a "set"
  - Duplicates removed

Joins

- Combining relations into one new relation
  - Many ways, variations
- \(<relation> \text{ CROSS JOIN } <relation>\)
  - Takes every possible combination

CROSS JOIN example

<table>
<thead>
<tr>
<th>VanTypes</th>
<th>SeatsAndPaint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
<td>Seats</td>
</tr>
<tr>
<td>Dodge</td>
<td>Cloth</td>
</tr>
<tr>
<td>Honda</td>
<td>Leather</td>
</tr>
<tr>
<td>Caravan</td>
<td>Standard</td>
</tr>
<tr>
<td>Odyssey</td>
<td>Premium</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Seats</th>
<th>Paint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dodge</td>
<td>Caravan</td>
<td>Cloth</td>
<td>Standard</td>
</tr>
<tr>
<td>Dodge</td>
<td>Caravan</td>
<td>Leather</td>
<td>Standard</td>
</tr>
<tr>
<td>Dodge</td>
<td>Caravan</td>
<td>Leather</td>
<td>Premium</td>
</tr>
<tr>
<td>Honda</td>
<td>Odyssey</td>
<td>Cloth</td>
<td>Standard</td>
</tr>
<tr>
<td>Honda</td>
<td>Odyssey</td>
<td>Leather</td>
<td>Standard</td>
</tr>
<tr>
<td>Honda</td>
<td>Odyssey</td>
<td>Leather</td>
<td>Premium</td>
</tr>
</tbody>
</table>

Inner Joins

- Inner Joins are based on the Cross Join
- Join is usually limited by some comparison using \(\text{ON} \) (Theta Join)
  
  e.g. \(\text{Senator \text{ INNER JOIN } Representative} \)
  \(\text{ON Senator.State = Representative.State}\)

  Creates table with one \((\text{Senator, Representative})\) tuple for every pair from the same state.
  
  (Note: \textit{both} State attributes still appear)
Natural Joins

- Automatically looks for matching columns
- Only one column for each match, and only select tuples that match in those columns

Natural Join Example

<table>
<thead>
<tr>
<th>Name</th>
<th>School</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe Smith</td>
<td>Rice</td>
<td>Houston</td>
</tr>
<tr>
<td>Jill Smith</td>
<td>LSU</td>
<td>Baton Rouge</td>
</tr>
<tr>
<td>Sam Jones</td>
<td>Texas A&amp;M</td>
<td>College Station</td>
</tr>
<tr>
<td>Sue Jones</td>
<td>Rice</td>
<td>Houston</td>
</tr>
</tbody>
</table>

OUTER JOIN

- Includes tuples from both relations, even if no match in the other
  - Those attributes are set to NULL
- LEFT, RIGHT, FULL
  - Keep all records from left table, or from right table, or from both