SQL: Data Manipulation Language

csc343, Introduction to Databases
Fatemeh Nargesian (slides are borrowed from Diane Horton)
Fall 2015
Introduction

• So far, we have defined database schemas and queries mathematically.
• SQL is a formal language for doing so with a DBMS.
• “Structured Query Language”, but it’s for more than writing queries.
• Two sub-parts:
  • DDL (Data Definition Language), for defining schemas.
  • DML (Data Manipulation Language), for writing queries and modifying the database.
PostgreSQL

• We’ll be working in PostgreSQL, an open-source relational DBMS.

• Learn your way around the documentation; it will be very helpful.

• Standards?
  • There are several. The most recent being SQL:2011.
  • The standards are not freely available. Must purchase from the International Standards Organization (ISO).
  • DBMSs vary in the details around the edges, making portability difficult.
A high-level language

• SQL is a very high-level language.
  • Say “what” rather than “how.”
• You write queries without manipulating data.
  • Contrasting languages like Java or C++.
• Provides physical “data independence:”
  • Details of how the data is stored can change with no impact on your queries.
• You can focus on readability.
  • But because the DMBS optimizes your query, you get efficiency.
Basic queries
[Slides 8-16 are essentially covered by Prep4]
Meaning of a query with one relation

SELECT name
FROM Course
WHERE dept = ‘CSC’;

\[ \pi_{\text{name}} (\sigma_{\text{dept}=”csc”} (\text{Course})) \]
SELECT vs $\sigma$

- **In SQL,**
  - SELECT is for choosing columns, i.e., $\pi$.
  - **Example:**
    ```sql
    select surName
    from Student
    where campus = 'StG';
    ```

- **In relational algebra,**
  - “select” means choosing rows, i.e., $\sigma$. 
Meaning of a query with one relation

```sql
SELECT name
FROM Course
WHERE dept = 'CSC';
```

\[
\pi_{\text{name}} (\sigma_{\text{dept} = 'csc'} (\text{Course}))
\]

\[
\left( \sigma_{\text{dept} = 'csc'} (\text{Course}) \right)
\]

\[
\text{(Course)}
\]
... and with multiple relations

```
SELECT name
FROM Course, Offering, Took
WHERE dept = 'CSC';
```

\[ \pi_{\text{name}} (\sigma_{\text{dept}="\text{csc}" } (\text{Course} \times \text{Offering} \times \text{Took})) \]
Temporarily renaming a table

• You can rename tables (just for the duration of the statement):

```
select e.name, d.name
from employee e, department d
where d.name = 'marketing'
and e.name = 'Horton';
```

• Can be convenient vs the longer full names:

```
select employee.name, department.name
from employee, department
where department.name = 'marketing'
and employee.name = 'Horton';
```

• This is like $\rho$ in relational algebra.
Self-joins

• As we know, renaming is required for self-joins.
• Example:

```sql
select e1.name, e2.name
from employee e1, employee e2
where e1.salary < e2.salary;
```
* In SELECT clauses

• A * in the SELECT clause means “all attributes of this relation.”

• Example:

  SELECT *
  FROM Course
  WHERE dept = ‘CSC’;
Renaming attributes

- Use `AS «new name»` to rename an attribute in the result.

- Example:
  ```sql
  SELECT name AS title, dept
  FROM Course
  WHERE breadth;
  ```
Complex Conditions in a WHERE

• We can build boolean expressions with operators that produce boolean results.
  • comparison operators: =, <> , <, >, <=, >=
  • and many other operators: see section 6.1.2 of the text book and chapter 9 of the postgreSQL documentation.

• We can combine boolean expressions with:
  • Boolean operators: AND, OR, NOT.
Example: Compound condition

- Find 3rd- and 4th-year CSC courses:

```
SELECT *
FROM Offering
WHERE dept = 'CSC' AND cNum >= 300;
```
ORDER BY

• To put the tuples in order, add this as the final clause:

  ORDER BY «attribute list» [DESC]

• The default is ascending order; DESC overrides it to force descending order.

• The attribute list can include expressions: e.g.,

  ORDER BY sales+rentals

• The ordering is the last thing done before the SELECT, so all attributes are still available.
ORDER BY

• SELECT *
  FROM Offering
  WHERE dept = 'CSC' AND cNum >= 300
ORDER BY cNum;
Case-sensitivity and whitespace

• Example query:
  
  ```
  select surName
  from Student
  where campus = 'StG';
  ```

• Keywords, like `select`, are not case-sensitive.
  • One convention is to use uppercase for keywords.

• Identifiers, like `Student` are not case-sensitive either.
  • One convention is to use lowercase for attributes, and a leading capital letter followed by lowercase for relations.

• Literal strings, like `'StG'`, are case-sensitive, and require single quotes.

• Whitespace (other than inside quotes) is ignored.
Expressions in SELECT clauses

• Instead of a simple attribute name, you can use an expression in a SELECT clause.
• Operands: attributes, constants
  Operators: arithmetic ops, string ops
• Examples:
  SELECT sid, grade-10 as adjusted
  FROM Took;

  SELECT dept||cnum
  FROM course;
Expressions that are a constant

• Sometimes it makes sense for the whole expression to be a constant (something that doesn’t involve any attributes!).

• Example:
  SELECT name,
    'satisfies' AS breadthRequirement
FROM Course
WHERE breadth;
Pattern operators

- Two ways to compare a string to a pattern by:
  - «attribute» LIKE «pattern»
  - «attribute» NOT LIKE «pattern»

- Pattern is a quoted string
  - % means: any string
  - _ means: any single character

- Example:
  ```sql
  SELECT *
  FROM Course
  WHERE name LIKE '%Comp%';
  ```
Aggregation
Computing on a column

• We often want to compute something across the values in a column.

• \texttt{SUM, AVG, COUNT, MIN,} and \texttt{MAX} can be applied to a column in a \texttt{SELECT} clause.

• Also, \texttt{COUNT( * )} counts the number of tuples.

• We call this aggregation.

• Note: To stop duplicates from contributing to the aggregation, use \texttt{DISTINCT} inside the brackets.
GROUP BY - Example

• How many courses (offerings) has each instructor offered?
• SELECT instructor, count(*)
  FROM Offerings
  GROUP BY instructor;

<table>
<thead>
<tr>
<th>instructor</th>
<th>count(*)</th>
<th>..</th>
</tr>
</thead>
</table>
| Ricky      | 2      | ...
| Smith      | 1      | ...
| Meyer      | 1      | ...

<table>
<thead>
<tr>
<th>oID</th>
<th>dept</th>
<th>cNum</th>
<th>term</th>
<th>instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>csc</td>
<td>343</td>
<td>201501</td>
<td>Ricky</td>
</tr>
<tr>
<td>o2</td>
<td>csc</td>
<td>180</td>
<td>201502</td>
<td>Rikcy</td>
</tr>
<tr>
<td>o3</td>
<td>csc</td>
<td>180</td>
<td>201502</td>
<td>Smith</td>
</tr>
<tr>
<td>o4</td>
<td>math</td>
<td>555</td>
<td>201501</td>
<td>Meyer</td>
</tr>
</tbody>
</table>
GROUP BY - Example

• How many courses (offerings) has each instructor offered?
  • SELECT instructor, count(*)
    FROM Offerings
    WHERE dept = 'csc'
    GROUP BY instructor;

26
Grouping

• If we follow a SELECT-FROM-WHERE expression with GROUP BY <attributes>
  • The tuples are grouped according to the values of those attributes, and
  • any aggregation is applied only within each group.
Restrictions on aggregation

- If any aggregation is used, then each element of the SELECT list must be either:
  - aggregated, or
  - an attribute on the GROUP BY list.
- Otherwise, it doesn’t even make sense to include the attribute.
HAVING Clauses

• WHERE let’s you decide which tuples to keep.
• Similarly, you can decide which groups to keep.
• Syntax:
  ...  
  GROUP BY «attributes»
  HAVING «condition»
• Semantics:
  Only groups satisfying the condition are kept.
HAVING - Example

- SELECT instructor, count(*)
  FROM Offerings
  WHERE dept = 'csc'
  GROUP BY instructor
  HAVING count(*) > 2;
Requirements on HAVING clauses

• Outside subqueries, HAVING may refer to attributes only if they are either:
  • aggregated, or
  • an attribute on the GROUP BY list.
• (The same requirement as for SELECT clauses with aggregation).
## Order of execution of a SQL query

<table>
<thead>
<tr>
<th>Query order</th>
<th>Execution order</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>FROM</td>
</tr>
<tr>
<td>FROM</td>
<td>WHERE</td>
</tr>
<tr>
<td>WHERE</td>
<td>GROUP BY</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>HAVING</td>
</tr>
<tr>
<td>HAVING</td>
<td>SELECT</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>ORDER BY</td>
</tr>
</tbody>
</table>
In-class Exercises!
Set operations
Tables can have duplicates in SQL

• A table can have duplicate tuples, unless this would violate an integrity constraint.
• And SELECT-FROM-WHERE statements leave duplicates in unless you say not to.
• Why?
  • Getting rid of duplicates is expensive!
  • We may want the duplicates because they tell us how many times something occurred.
Relational Algebra with Bags

• **Reference:** section 5.1 of the text.

• **Behaviour of most operations is no different.**
  - $\sigma$, $\rho$: as before
  - $\pi$: duplicates are not removed.
  - joins: duplicates can proliferate
Bags

- SQL treats tables as “bags” (or “multisets”) rather than sets.
- Bags are just like sets, but duplicates are allowed.
- \{6, 2, 7, 1, 9\} is a set (and a bag)
  \{6, 2, 2, 7, 1, 9\} is not a set, but is a bag.
- Like with sets, order doesn’t matter.
  \{6, 2, 7, 1, 9\} = \{1, 2, 6, 7, 9\}
Union, Intersection, and Difference

- These are expressed as:
  
  \[(\text{«subquery»}) \text{ UNION } (\text{«subquery»})\]
  
  \[(\text{«subquery»}) \text{ INTERSECT } (\text{«subquery»})\]
  
  \[(\text{«subquery»}) \text{ EXCEPT } (\text{«subquery»})\]

- The brackets are mandatory.

- The operands must be queries; you can’t simply use a relation name.
Example

(SELECT sid
 FROM Took
 WHERE grade > 95)
 UNION

(SELECT sid
 FROM Took
 WHERE grade < 50);

Schemas should match!
In-class Exercises!
Operations $\cup$, $\cap$, and $-$ with Bags

• For $\cup$, $\cap$, and $-$ the number of occurrences of a tuple in the result requires some thought.

• Exercises:
  1. $\{1, 1, 1, 3, 7, 7, 8\} \cup \{1, 5, 7, 7, 8, 8\}$
  2. $\{1, 1, 1, 3, 7, 7, 8\} \cap \{1, 5, 7, 7, 8, 8\}$
  3. $\{1, 1, 1, 3, 7, 7, 8\} - \{1, 5, 7, 7, 8, 8\}$
1. \{1, 1, 1, 3, 7, 7, 8\} \cup \{1, 5, 7, 7, 8, 8\} \\
\quad = \{1, 1, 1, 3, 7, 7, 8, 1, 5, 7, 7, 8, 8\} \\
\quad = \{1, 1, 1, 1, 3, 5, 7, 7, 7, 8, 8, 8\} \\
2. \{1, 1, 1, 3, 7, 7, 8\} \cap \{1, 5, 7, 7, 8, 8\} \\
\quad = \{1, 7, 7, 8\} \\
3. \{1, 1, 1, 3, 7, 7, 8\} \setminus \{1, 5, 7, 7, 8, 8\} \\
\quad = \{1, 1, 3\}
Operations $\cup$, $\cap$, and $-$ with Bags

- Suppose tuple $t$ occurs
  - $m$ times in relation $R$, and
  - $n$ times in relation $S$.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of occurrences of $t$ in result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R \cap S$</td>
<td>$\min(m, n)$</td>
</tr>
<tr>
<td>$R \cup S$</td>
<td>$m + n$</td>
</tr>
<tr>
<td>$R - S$</td>
<td>$\max(m-n, 0)$</td>
</tr>
</tbody>
</table>
Bag vs Set Semantics: which is used

• We saw that a SELECT-FROM-WHERE statement uses bag semantics by default.
  • Duplicates are kept in the result.
• The set operations use set semantics by default.
  • Duplicates are eliminated from the result.
Motivation: Efficiency

• When doing projection, it is easier not to eliminate duplicates.
  • Just work one tuple at a time.

• For intersection or difference, it is most efficient to sort the relations first.
  • At that point you may as well eliminate the duplicates anyway.
Controlling Duplicate Elimination

- We can force the result of a SFW query to be a set by using `SELECT DISTINCT ...`.
  
  ```sql
  SELECT DISTINCT sid
  FROM Took
  WHERE grade > 95
  ```

- We can force the result of a set operation to be a bag by using `ALL`, e.g.,
  
  ```sql
  (SELECT sid
   FROM Took
   WHERE grade > 95)
  UNION ALL
  (SELECT sid
   FROM Took
   WHERE grade > 95)
  ```
Views
The idea

• A view is a relation defined in terms of stored tables (called base tables) and other views.
• Access a view like any base table.
• Two kinds of view:
  • **Virtual**: no tuples are stored; view is just a query for constructing the relation when needed.
  • **Materialized**: actually constructed and stored. Expensive to maintain!
• We’ll use only virtual views.
  • PostgreSQL did not support materialized views until version 9.3 (which we are not running).
Example: defining a virtual view

- A view for students who earned an 80 or higher in a CSC course.

```
CREATE VIEW topresults as
SELECT firstname, surname, cnum
FROM Student, Took, Offering
WHERE
    Student.sid = Took.sid AND
    Took.oid = Offering.oid AND
    grade >= 80 AND dept = 'CSC';
```
Uses for views

• Break down a large query.
• Provide another way of looking at the same data, e.g., for one category of user.
In-class Exercises!