Lecture 02: SQL

Friday, September 29, 2006
Administrivia

- Homework 1 is out. Due: Wed., Oct. 11
- Did you login on IISQLSRV?
- Did you change your password?
- Did you subscribe to CSE444?
Today’s Reading Assignment

• Did you read it?

• What does ACID mean?
  
  A = atomicity
  
  C = consistency
  
  I = isolation
  
  D = durability
Outline

• Data in SQL
• Simple Queries in SQL (6.1)
• Queries with more than one relation (6.2)
SQL Introduction

Standard language for querying and manipulating data

Structured Query Language

Many standards out there:
- ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
- Vendors support various subsets: watch for fun discussions in class!
SQL

• Data Definition Language (DDL)
  – Create/alter/delete tables and their attributes
  – Following lectures...

• Data Manipulation Language (DML)
  – Query one or more tables – discussed next!
  – Insert/delete/modify tuples in tables
# Tables in SQL

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Tables Explained

• The *schema* of a table is the table name and its attributes:

  Product(PName, Price, Category, Manufacturer)

• A *key* is an attribute whose values are unique; we underline a key

  Product(PName, Price, Category, Manufacturer)
Data Types in SQL

- Atomic types:
  - Characters: CHAR(20), VARCHAR(50)
  - Numbers: INT, BIGINT, SMALLINT, FLOAT
  - Others: MONEY, DATETIME, …

- Every attribute must have an atomic type
  - Hence tables are flat
  - Why?
Tables Explained

• A tuple = a record
  – Restriction: all attributes are of atomic type

• A table = a set of tuples
  – Like a list…
  – …but it is unordered:
    no first(), no next(), no last().
SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>
```
Simple SQL Query

```
SELECT * FROM Product WHERE category='Gadgets'
```

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

“selection”
Simple SQL Query

```
SELECT PName, Price, Manufacturer FROM Product WHERE Price > 100
```

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

“selection” and “projection”
Notation

Input Schema

Product(PName, Price, Category, Manufacturer)

\[
\begin{align*}
\text{SELECT} & \quad \text{PName, Price, Manufacturer} \\
\text{FROM} & \quad \text{Product} \\
\text{WHERE} & \quad \text{Price} > 100
\end{align*}
\]

Answer(PName, Price, Manufacturer)

Output Schema
Details

• Case insensitive:
  – Same: SELECT  Select  select
  – Same: Product  product
  – Different: ‘Seattle’ ‘seattle’

• Constants:
  – ‘abc’ - yes
  – “abc” - no
The **LIKE** operator

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character
Eliminating Duplicates

```
SELECT DISTINCT category
FROM Product
```

Compare to:

```
SELECT category
FROM Product
```

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Photography</td>
</tr>
<tr>
<td>Household</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Photography</td>
</tr>
<tr>
<td>Household</td>
</tr>
</tbody>
</table>

Ordering the Results

```
SELECT  pname, price, manufacturer  
FROM    Product  
WHERE   category='gizmo' AND price > 50  
ORDER BY price, pname
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**SELECT DISTINCT** category
FROM Product
ORDER BY category

**SELECT** Category
FROM Product
ORDER BY PName

**SELECT DISTINCT** category
FROM Product
ORDER BY PName
### Company

<table>
<thead>
<tr>
<th>CName</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price FROM Product, Company WHERE Manufacturer=CName AND Country='Japan' AND Price <= 200
```
Joins

**SELECT** PName, Price  
**FROM** Product, Company  
**WHERE** Manufacturer=CName AND Country='Japan' AND Price <= 200

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cname</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

SingleTouch $149.99
More Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all Chinese companies that manufacture products both in the ‘electronic’ and ‘toy’ categories

```
SELECT    cname

FROM

WHERE
```
A Subtlety about Joins

Product (pname,  price, category, manufacturer)
Company (cname, stockPrice, country)

Find all countries that manufacture some product in the ‘Gadgets’ category.

```
SELECT Country
FROM Product, Company
WHERE Manufacturer=CName AND Category='Gadgets'
```
### A Subtlety about Joins

#### What is the problem? What’s the solution?

```sql
SELECT Country
FROM Product, Company
WHERE Manufacturer=CName AND Category='Gadgets'
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cname</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

#### SELECT Country
FROM Product, Company
WHERE Manufacturer=CName AND Category='Gadgets'
```
Tuple Variables

Person\(\text{pname, address, worksfor}\)
Company\(\text{cname, address}\)

\[
\text{SELECT DISTINCT } \text{pname, address} \\
\text{FROM } \text{Person, Company} \\
\text{WHERE worksfor = cname}
\]

Which address?

\[
\text{SELECT DISTINCT } \text{Person.pname, Company.address} \\
\text{FROM } \text{Person, Company} \\
\text{WHERE } \text{Person.worksfor = Company.cname}
\]

\[
\text{SELECT DISTINCT } x.\text{pname, y.address} \\
\text{FROM } \text{Person AS x, Company AS y} \\
\text{WHERE } x.\text{worksfor = y.cname}
\]
Meaning (Semantics) of SQL Queries

\[
\text{SELECT } a_1, a_2, \ldots, a_k \\
\text{FROM } R_1 \text{ AS } x_1, R_2 \text{ AS } x_2, \ldots, R_n \text{ AS } x_n \\
\text{WHERE Conditions}
\]

\[
\text{Answer} = \{\} \\
\text{for } x_1 \text{ in } R_1 \text{ do} \\
\quad \text{for } x_2 \text{ in } R_2 \text{ do} \\
\quad \quad \text{.....} \\
\quad \quad \text{for } x_n \text{ in } R_n \text{ do} \\
\quad \quad \quad \text{if Conditions} \\
\quad \quad \quad \quad \text{then Answer} = \text{Answer} \cup \{ (a_1, \ldots, a_k) \} \\
\text{return Answer}
\]
An Unintuitive Query

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A  OR  R.A=T.A
```

What does it compute?

Computes \( R \cap (S \cup T) \)  

But what if \( S = \emptyset \)?
Subqueries Returning Relations

Company(\textit{name}, \textit{city})
Product(\textit{pname}, \textit{maker})
Purchase(\textit{id}, \textit{product}, \textit{buyer})

Return cities where one can find companies that manufacture products bought by Joe Blow

\begin{verbatim}
SELECT Company.city FROM Company WHERE Company.name IN (SELECT Product.maker FROM Purchase , Product WHERE Product.pname=Purchase.product AND Purchase .buyer = 'Joe Blow');
\end{verbatim}
Subqueries Returning Relations

Is it equivalent to this?

```sql
SELECT Company.city
FROM Company, Product, Purchase
WHERE Company.name = Product.maker
  AND Product.pname = Purchase.product
  AND Purchase.buyer = 'Joe Blow'
```

Beware of duplicates!
Removing Duplicates

```sql
SELECT DISTINCT Company.city
FROM Company
WHERE Company.name IN
    (SELECT Product.maker
     FROM Purchase, Product
     WHERE Product.pname = Purchase.product
     AND Purchase.buyer = 'Joe Blow');
```

Now they are equivalent

```sql
SELECT DISTINCT Company.city
FROM Company, Product, Purchase
WHERE Company.name = Product.maker
     AND Product.pname = Purchase.product
     AND Purchase.buyer = 'Joe Blow'
```
Subqueries Returning Relations

You can also use: \( s > \text{ALL} R \)
\( s > \text{ANY} R \)
\( \exists R \)

Product ( pname, price, category, maker)
Find products that are more expensive than all those produced By “Gizmo-Works”

```
SELECT name
FROM Product
WHERE price > ALL (SELECT price
                    FROM Product
                    WHERE maker='Gizmo-Works')
```
Question for Database Fans and their Friends

• Can we express this query as a single SELECT-FROM-WHERE query, without subqueries?

• Answer: all SFW queries are monotone (figure out what this means). A query with ALL is not monotone
Correlated Queries

Movie (title, year, director, length)
Find movies whose title appears more than once.

Note (1) scope of variables (2) this can still be expressed as single SFW
Complex Correlated Query

Product (pname, price, category, maker, year)

• Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT DISTINCT pname, maker
FROM Product AS x
WHERE price > ALL (SELECT price
                   FROM Product AS y
                   WHERE x.maker = y.maker AND y.year < 1972);
```

Very powerful! Also much harder to optimize.
Reading Assignment for Monday

SQL from the textbook:
• Renaming columns: SELECT x.name AS nom
• Union, intersection, difference

Chapter 3, “Simple Queries” from SQL for Web Nerds, by Philip Greenspun
http://philip.greenspun.com/sql/