Lecture 02: SQL

Wednesday, March 28, 2007
Administrivia

- Homework 1 is out. Due: Fri., April 6
- Did you login on IPROJSRV?
- Did you change your password?
- Did you subscribe to CSE444?
- Did you read today’s reading assignment?
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 Name

<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>

### Attribute Names

- **PName**: Product Name
- **Price**: Price
- **Category**: Category
- **Manufacturer**: Manufacturer
Tables Explained

- The *schema* of a table is the table name and its attributes:
  \[ \text{Product}(\text{PName}, \text{Price}, \text{Category}, \text{Manfacturer}) \]

- A *key* is an attribute whose values are unique; we underline a key:
  \[ \text{Product}(\text{PName}, \text{Price}, \text{Category}, \text{Manfacturer}) \]
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)

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

```sql
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>
Simple SQL Query

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

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

“selection” and “projection”
Notation

Input Schema

Product(PName, Price, Category, Manufacturer)

Output Schema

Answer(PName, Price, Manufacturer)

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

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

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

```sql
SELECT * FROM Products WHERE PName LIKE '%gizmo%'
```

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

```sql
SELECT DISTINCT category
FROM   Product
```

Compare to:

```sql
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
Keys and Foreign Keys

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
```

Join between Product and Company
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>

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
</tr>
</tbody>
</table>
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

**Product**

<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>

**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>

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

What is the problem?
A Subtlety about Joins

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pownergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</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>
<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>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

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

Duplicates! What’s the solution?
Tuple Variables

Person(pname, address, worksfor)
Company(cname, address)

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

Which address?

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

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

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

Answer = {} \\
for \(x_1\) in \(R_1\) do \\
    for \(x_2\) in \(R_2\) do \\
        \ldots\ \\
        for \(x_n\) in \(R_n\) do \\
            if \text{Conditions} \\
                then Answer = Answer \cup \{(a_1, \ldots, a_k)\}

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(name, city)
Product(pname, maker)
Purchase(id, product, buyer)

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

```
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');
```
Subqueries Returning Relations

Is it equivalent to this?

```
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:  
\[ s > \text{ALL R} \]
\[ s > \text{ANY R} \]
\[ \text{EXISTS R} \]

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

```sql
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?
Monotone Queries

• A query Q is monotone if:
  – Whenever we add tuples to one or more of the tables…
  – … the answer to the query cannot contain fewer tuples

• Fact: all SFW (select-from-where) queries are monotone

• Fact: A query with ALL is not monotone

• Consequence: we cannot rewrite an ALL query into a SFW
Correlated Queries

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

```
SELECT DISTINCT title 
FROM Movie AS x 
WHERE year <> ANY 
    (SELECT year 
     FROM Movie 
     WHERE title = x.title);
```

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.