Lecture 02: SQL

Friday, January 6, 2006

Administrivia

• Homework 1 is out. Due: Wed., Jan. 18

• 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>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>

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

<table>
<thead>
<tr>
<th>Product</th>
<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>
<td></td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
<td></td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
<td></td>
</tr>
</tbody>
</table>

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

```
<table>
<thead>
<tr>
<th>Product</th>
<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>
<td></td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
</tbody>
</table>
```

“selection”
Simple SQL Query

```
<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>Powegizmo</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 PName, Price, Manufacturer
FROM Product
WHERE Price > 100
```

“selection” and “projection”

Notation

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

Input Schema

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

```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

```
SELECT DISTINCT category
FROM Product
```

Compare to:

```
SELECT category
FROM Product
```

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.
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
```
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

<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>Gizmo/Works</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td></td>
<td>Gizmo/Works</td>
</tr>
<tr>
<td>Single/Touch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Multi/Touch</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>Gizmo/Works</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? What’s the solution?

Tuple Variables

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

SELECT DISTINCT pname, address
FROM Person, Company
WHERE worksfor = cname

SELECT DISTINCT Person.pname, Company.address
FROM Person, Company
WHERE Person.worksfor = Company.cname

SELECT DISTINCT x.pname, y.address
FROM Person AS x, Company AS y
WHERE x.worksfor = y.cname
Meaning (Semantics) of SQL Queries

```sql
SELECT a1, a2, ..., ak
FROM R1 AS x1, R2 AS x2, ..., Rn AS xn
WHERE Conditions
```

```python
Answer = {}
for x1 in R1 do
    for x2 in R2 do
        ...
        for xn in Rn do
            if Conditions then
                Answer = Answer ∪ {(a1, ..., ak)}
        return Answer
```

An Unintuitive Query

```sql
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 $S = \varnothing$?
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

```sql
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');
```

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');
```

```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 Purchase
                        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?

• Hint: show that 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.

```
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

```sql
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/