Introduction to Database Systems
CSE 444

Lecture 02: SQL
April 2, 2008

Administrivia

• Homework 1 is out. Due: Fri., April 11
• Did you login on IISQLSRV?
• Did you change your password?
• Did you read today’s reading assignment?
  – (Do you remember what it was?)
• Project 0: Who’s your partner?
  – Due next Wednesday

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'
```

```
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)

**Details**

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

- Constants:
  - ‘abc’ - yes
  - “abc” - no

**The LIKE operator**

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

**Eliminating Duplicates**

- `DISTINCT` category

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

<table>
<thead>
<tr>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>CName</td>
</tr>
<tr>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Canon</td>
</tr>
<tr>
<td>Hitachi</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

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

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

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

More Joins

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

SELECT Cname
FROM Company
WHERE Country='China'
AND CName IN (SELECT Manufacturer FROM Product WHERE Category IN ('electronic', 'toy'))

A Subtlety about Joins

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

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

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

What is the problem?

Country

???
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>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>

Duplicates! What's the solution?

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

Tuple Variables

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

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

Which address?

```
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
```

Which address?

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

Meaning (Semantics) of SQL Queries

```
SELECT a_1, a_2, ..., a_k
FROM R_1 AS x_1, R_2 AS x_2, ..., R_n AS x_n
WHERE Conditions
Answer = {}
for x_1 in R_1 do
  for x_2 in R_2 do
    ....
    for x_k in R_k do
      if Conditions
        then Answer = Answer ∪ {(a_1, ..., 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
```

Computes $R \cap (S \cup T)$ But what if $S = \emptyset$?
Subqueries Returning Relations

Company(name, city)
Product(name, 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

```
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');
```

Subqueries Returning Relations

Now they are equivalent

```
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 > ALL R
s > 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?

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.