Basic SQL

Lecture 2

Outline

• Data in SQL
• Simple Queries in SQL
• Queries with more than one relation

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

Standard language for querying and manipulating data

Structured Query Language

Many open standards out there:
- ANSI SQL
- SQL89 (a.k.a. SQL1)
- SQL92 (a.k.a. SQL2)
- SQL99 (a.k.a. SQL3)
- SQL:2003

Vendors support various subsets of these
What we discuss is common to all of them

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
- Transact-SQL
  - Idea: package a sequence of SQL statements → server
  - Won’t discuss in class
Data in SQL

1. Atomic types, a.k.a. data types
2. Tables built from atomic types

Data Types in SQL

• Characters:
  – CHAR(20) -- fixed length
  – VARCHAR(40) -- variable length

• Numbers:
  – BIGINT, INT, SMALLINT, TINYINT
  – REAL, FLOAT -- differ in precision
  – MONEY

• Times and dates:
  – DATE
  – DATETIME -- SQL Server

• Others... All are simple
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>

Table Details

- 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()`.
- No nested tables, only flat tables are allowed!
  - We will see later how to decompose complex structures into multiple flat tables
Table Details

• The *schema* of a table is the table name and its attributes:
  Product(PName, Price, Category, Manfacturer)

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

  Product(PName, Price, Category, Manfacturer)

SQL Query

Basic form:

```
SELECT attributes
FROM relations (possibly multiple, joined)
WHERE conditions (selections)
```
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>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>
</tbody>
</table>
```

Simple SQL Query

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

```
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
```
A Notation for SQL Queries

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

Input Schema

Product(PName, Price, Category, Manufacturer)

Answer(PName, Price, Manufacturer)

Output Schema

Selections

What goes in the WHERE clause:

- $x = y$, $x < y$, $x \leq y$, etc
  - For number, they have the usual meanings
  - For CHAR and VARCHAR: lexicographic ordering
    - Expected conversion between CHAR and VARCHAR
  - For dates and times, what you expect...
- Pattern matching on strings: s LIKE p (next)
The LIKE operator

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

Product(Name, Price, Category, Manufacturer)
Find all products whose name mentions 'gizmo':

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

Eliminating Duplicates

```
SELECT category FROM Product
```

Compare to:

```
SELECT DISTINCT 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>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
```

Ordering is **ascending**, unless you specify the DESC keyword.

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

---

Ordering the Results

```
SELECT Category
FROM Product
ORDER BY PName
```

<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>
**Ordering the Results**

```
SELECT DISTINCT category
FROM Product
ORDER BY category
```

Compare to:

```
SELECT DISTINCT category
FROM Product
ORDER BY PName
```

---

**Joins in SQL**

- Connect two or more tables:

<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>

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

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

Joins in SQL

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>PName</td>
<td>CName</td>
</tr>
<tr>
<td>Price</td>
<td>StockPrice</td>
</tr>
<tr>
<td>Category</td>
<td>Country</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>$19.99</td>
<td>25</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>Gadgets</td>
</tr>
<tr>
<td>$29.99</td>
<td>65</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>Photography</td>
</tr>
<tr>
<td>$149.99</td>
<td>15</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>Household</td>
</tr>
<tr>
<td>$203.99</td>
<td>10</td>
</tr>
</tbody>
</table>

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

Product (pname, price, category, manufacturer)
Purchase (buyer, seller, store, product)
Person(persname, phoneNumber, city)

Find names of people living in Seattle that bought some product in the ‘Gadgets’ category, and the names of the stores they bought such product from

```sql
SELECT DISTINCT persname, store
FROM Person, Purchase, Product
WHERE persname=buyer AND product = pname AND
   city='Seattle' AND category='Gadgets'
```

Disambiguating Attributes

- Sometimes two relations have the same attr:
  Person(pname, address, worksfor)
  Company(cname, address)

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

Which address?

```sql
SELECT DISTINCT Person.pname, Company.address
FROM Person, Company
WHERE Person.worksfor = Company.cname
```
**Tuple Variables**

Product (pname, price, category, manufacturer)
Purchase (buyer, seller, store, product)
Person(persname, phoneNumber, city)

Find all stores that sold at least one product that the store ‘BestBuy’ also sold:

```
SELECT DISTINCT x.store
FROM Purchase AS x, Purchase AS y
WHERE x.product = y.product AND y.store = ‘BestBuy’
```

Answer (store)

---

**Tuple Variables**

General rule:
tuple variables introduced automatically by the system:

Product (name, price, category, manufacturer)

```
SELECT name
FROM Product
WHERE price > 100
```

Becomes:

```
SELECT Product.name
FROM Product AS Product
WHERE Product.price > 100
```

 Doesn’t work when Product occurs more than once:
In that case the user needs to define variables explicitly.
Renaming Columns

<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 Pname AS prodName, Price AS askPrice
FROM Product
WHERE Price > 100

Query with renaming

<table>
<thead>
<tr>
<th>prodName</th>
<th>askPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
</tr>
</tbody>
</table>

Meaning (Semantics) of SQL Queries

SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions

1. Nested loops:

Answer = {}
for x1 in R1 do
    for x2 in R2 do
        …
            for xn in Rn do
                if Conditions
                    then Answer = Answer ∪ {(x1,…,xk)}
    return Answer
Meaning (Semantics) of SQL Queries

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

2. Parallel assignment

Answer = {}
for all assignments x1 in R1, …, xn in Rn do
   if Conditions then Answer = Answer ∪ {(x1,…,xk)}
return Answer

Doesn’t impose any order!

First Unintuitive SQLism

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

Looking for R ∩ (S ∪ T)

But what happens if T is empty?
SELECT Question
FROM Student
WHERE Course=“IIS”