Lecture 02: SQL

Wednesday, March 31st, 2010
Accessing SQL Server

• Host: IISQLSRV.cs.washington.edu

• Authentication: SQL Server Authentication

• User: YOUR_USER_NAME@u.washington.edu

• Password: 'cse444login!' (without the quotes)

• Change your password!
Outline

• Data in SQL
• Simple Queries in SQL (6.1)
• Queries with more than one relation (6.2)
• Subqueries (6.3)
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>

**Product**: Key

**Table name**: Tuples or rows

**Attribute names**: Dan Suciu -- 444 Spring 2010
Data Types in SQL

• Atomic types:
  – Characters: CHAR(20), VARCHAR(50)
  – Numbers: INT, BIGINT, SMALLINT, FLOAT
  – Others: MONEY, DATETIME, ...

• Record (aka tuple)
  – Has atomic attributes

• Table (relation)
  – A set of tuples
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

### 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>

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

“selection” and “projection”
Details

• Case insensitive:
  
  SELECT = Select = select
  
  Product = product
  
  BUT: ‘Seattle’ ≠ ‘seattle’

• Constants:
  
  ‘abc’ - yes
  
  “abc” - no
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>
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.

Ordering is ascending, unless you specify the DESC keyword.
### SQL Queries

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

2. **SELECT** Category 
   FROM Product 
   ORDER BY PName

3. **SELECT DISTINCT** category 
   FROM Product 
   ORDER BY PName

### Table

<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>
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.

\[
\text{SELECT PName, Price FROM Product, Company WHERE Manufacturer=CName AND Country='Japan' AND Price \leq 200}
\]
## Joins

<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>

**Query:**

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

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

<table>
<thead>
<tr>
<th>SQL Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{SELECT DISTINCT } \text{pname, address}$</td>
</tr>
<tr>
<td>$\text{FROM } \text{Person, Company}$</td>
</tr>
<tr>
<td>$\text{WHERE worksfor = cname}$</td>
</tr>
<tr>
<td>$\text{SELECT DISTINCT Person.pname, Company.address}$</td>
</tr>
<tr>
<td>$\text{FROM } \text{Person, Company}$</td>
</tr>
<tr>
<td>$\text{WHERE Person.worksfor = Company.cname}$</td>
</tr>
<tr>
<td>$\text{SELECT DISTINCT x.pname, y.address}$</td>
</tr>
<tr>
<td>$\text{FROM } \text{Person AS x, Company AS y}$</td>
</tr>
<tr>
<td>$\text{WHERE x.worksfor = y.cname}$</td>
</tr>
</tbody>
</table>

---

Which address?
In Class

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

Find all Chinese companies that manufacture products both in the ‘toy’ category

SELECT  cname
FROM
WHERE
In Class

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     
```
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_n in R_n do
      if Conditions
        then Answer = Answer \cup \{(a_1,\ldots,a_k)\}
return Answer
```
Using the Formal Semantics

What do these queries compute?

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

Returns \(R \cap S\)

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

If \(S \neq \emptyset\) and \(T \neq \emptyset\) then returns \(R \cap (S \cup T)\)
else returns \(\emptyset\)
Joins Introduce Duplicates

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 Introduce Duplicates

### 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>

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

Duplicates! Remember to add DISTINCT

Dan Suciu -- 444 Spring 2010
Subqueries

• A subquery is another SQL query nested inside a larger query
• Such inner-outer queries are called nested queries
• A subquery may occur in:
  1. A SELECT clause
  2. A FROM clause
  3. A WHERE clause

Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it’s impossible
1. Subqueries in SELECT

Product (pname, price, company)
Company(cname, city)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city
              FROM Company Y
              WHERE Y.cname=X.company)
FROM Product X
```

What happens if the subquery returns more than one city?
1. Subqueries in SELECT

Product (pname, price, company)
Company(cname, city)

Whenever possible, don’t use a nested queries:

\[
\text{SELECT pname, (SELECT city FROM Company WHERE cname=company) FROM Product}
\]

\[
\text{SELECT pname, city FROM Product, Company WHERE cname=company}
\]

We have “unnested” the query
1. Subqueries in SELECT

Product (\texttt{pname}, \texttt{price}, \texttt{company})

Company(\texttt{cname}, \texttt{city})

Compute the number of products made in each city

\begin{verbatim}
SELECT DISTINCT city, (SELECT count(*) FROM Product WHERE cname=company)
FROM Company
\end{verbatim}

Better: we can unnest by using a GROUP BY (next lecture)
2. Subqueries in FROM

Product ( pname, price, company)
Company (cname, city)

Find all products whose prices is > 20 and < 30

SELECT X.city
FROM (SELECT * FROM Product AS Y WHERE Y.price > 20) AS X
WHERE X.price < 30

Unnest this query!
3. Subqueries in WHERE

Product ( pname, price, company)  
Company( cname, city)

Find all cities that make some products with price < 100

Using EXISTS:

SELECT DISTINCT Company.city
FROM Company
WHERE EXISTS (SELECT * 
FROM Product
WHERE company = cname and Produc.price < 100)
3. Subqueries in WHERE

Product ( pname, price, company)  
Company( cname, city)

Find all cities that make some products with price < 100

Predicate Calculus (a.k.a. First Order Logic)

{ y | ∃ x. Company(x,y) ∧ (∃ z. ∃ p. Product(z,p,x) ∧ p < 100) }
3. Subqueries in WHERE

Product (pname, price, company)  
Company (cname, city)

Find all cities that make some products with price < 100

Using IN

```
SELECT DISTINCT Company.city
FROM Company
WHERE Company.cname IN (SELECT Product.company
                          FROM Product
                          WHERE Product.price < 100)
```
3. Subqueries in WHERE

Product ( pname, price, company)  Existential quantifiers
Company( cname, city)

Find all cities that make some products with price < 100

Using ANY:

```
SELECT DISTINCT Company.city
FROM Company
WHERE 100 > ANY (SELECT price
FROM Product
WHERE company = cname)
```
3. Subqueries in WHERE

Product ( \textit{pname}, \textit{price}, \textit{company})
Company( \textit{cname}, \textit{city})

Find all cities that make some products with price < 100

Now let’s unnest it:

\begin{verbatim}
SELECT DISTINCT Company.cname
FROM Company, Product
WHERE Company.cname = Product.company and Product.price < 100
\end{verbatim}

Existential quantifiers are easy!
3. Subqueries in WHERE

Product (pname, price, company)
Company(cname, city)

Find all cities with companies that make only products with price < 100

Universal quantifiers are hard! 😞
3. Subqueries in WHERE

Product (pname, price, company)

Company(cname, city)

Find all cities with companies that make only products with price < 100

Predicate Calculus (a.k.a. First Order Logic)

\{ y | \exists x. \text{Company}(x,y) \land (\forall z. \forall p. \text{Product}(z,p,x) \implies p < 100) \}
3. Subqueries in WHERE

De Morgan’s Laws:

\[
\neg (A \land B) = \neg A \lor \neg B \\
\neg (A \lor B) = \neg A \land \neg B \\
\neg \forall x. \ P(x) = \exists x. \ \neg P(x) \\
\neg \exists x. \ P(x) = \forall x. \ \neg P(x)
\]

\[
\{ y | \exists x. \ \text{Company}(x,y) \land (\forall z. \ \forall p. \ \text{Product}(z,p,x) \Rightarrow p < 100) \} = \\
\{ y | \exists x. \ \text{Company}(x,y) \land \neg (\exists z \exists p. \ \text{Product}(z,p,x) \land p \geq 100) \} = \\
\{ y | \exists x. \ \text{Company}(x,y) \} - \{ y | \exists x. \ \text{Company}(x,y) \land (\exists z \exists p. \ \text{Product}(z,p,x) \land p \geq 100) \}
\]
3. Subqueries in WHERE

1. Find *the other* companies: i.e. s.t. *some* product \(\geq 100\)

   ```sql
   SELECT DISTINCT Company.city
   FROM Company
   WHERE Company.cname IN (SELECT Product.company
                             FROM Product
                             WHERE Product.price >= 100)
   ```

2. Find all companies s.t. *all* their products have price \(< 100\)

   ```sql
   SELECT DISTINCT Company.city
   FROM Company
   WHERE Company.cname NOT IN (SELECT Product.company
                                 FROM Product
                                 WHERE Product.price >= 100)
   ```
3. Subqueries in WHERE

Product (pname, price, company)
Company (cname, city)

Find all cities with companies that make only products with price < 100

Using EXISTS:

```
SELECT DISTINCT Company.city
FROM Company
WHERE NOT EXISTS (SELECT * FROM Product
WHERE company = cname and Product.price >= 100)
```
3. Subqueries in WHERE

Product (pname, price, company)  
Company(cname, city)

Find all cities that make some products with price < 100

Using ALL:

```
SELECT DISTINCT Company.city  
FROM Company  
WHERE 100 > ALL (SELECT price  
FROM Product  
WHERE company = cname)
```
Question for Database Fans and their Friends

• Can we unnest the *universal quantifier* query?
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 unnested queries are monotone
  – Proof: using the “nested for loops” semantics

• **Fact**: A query a universal quantifier is not monotone

• **Consequence**: we cannot unnest a query with a universal quantifier
Queries that must be nested

• Queries with universal quantifiers or with negation
• The drinkers-bars-beers example next
• This is a famous example from textbook on databases by Ullman
The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Challenge: write these in SQL

Find drinkers that frequent some bar that serves some beer they like.

\[ x: \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent only bars that serves some beer they like.

\[ x: \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]

Find drinkers that frequent some bar that serves only beers they like.

\[ x: \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]

Find drinkers that frequent only bars that serves only beer they like.

\[ x: \forall y. \text{Frequents}(x, y) \Rightarrow \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]