Introduction to Database Systems
CSE 444

Lecture 2: SQL
Announcements

• Project 1 is posted on class website
  – Due in two weeks (11 pm)
  – Remember: time goes by very fast! Start early!

• Have you logged in to the database yet?
  – If not, better do it now and let us know if there are any problems!
Outline

• Data in SQL
• Simple Queries in SQL (6.1)
• Queries with more than one relation (6.2)
• Subqueries (6.3)
Structured Query Language (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><strong>PName</strong></th>
<th><strong>Price</strong></th>
<th><strong>Category</strong></th>
<th><strong>Manufacturer</strong></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>
Data Types in SQL

• **Atomic types**
  – Character strings: CHAR(20), VARCHAR(50)
    • Can be of fixed or variable length
  – Numbers: INT, BIGINT, SMALLINT, FLOAT
  – Others: MONEY, DATETIME, …

• **Record (aka tuple)**
  – Has atomic attributes

• **Table (relation)**
  – A set of tuples

Book Sec. 2.3.2
## 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>

**SELECT**

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

---

**“selection”**

<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>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" and "projection"
Details

• **SQL is case insensitive**
  - SELECT = Select = select
  - Product = product
  - BUT ‘Seattle’ ≠ ‘seattle’ (in general)

• **Constants must use single quotes**
  - ‘abc’ - yes
  - “abc” - no
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>

CSE 444 - Summer 2009
Ordering the Results

```
SELECT   pname, price, manufacturer
FROM      Product
WHERE     category='gadgets' AND price > 10
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>

**SQL Query 1**

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

**SQL Query 2**

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

**SQL Query 3**

```sql
SELECT DISTINCT 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>

SELECT DISTINCT category
FROM Product
ORDER BY category

SELECT Category
FROM Product
ORDER BY PName

SELECT DISTINCT category
FROM Product
ORDER BY PName

Category
- Gadgets
- Household
- Photography

Error
# 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
```
Joins

### 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>

### 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** PName, Price  
**FROM** Product, Company  
**WHERE** Manufacturer=CName AND Country='Japan'  
AND Price <= 200
In Class

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

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

```
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'China' AND category = 'toy' AND manufacturer = cname
```
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

```sql
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 DISTINCT cname
FROM Product P1, Product P2, Company
WHERE country = 'China' AND P1.category = 'toy'
AND P2.category = 'electronic' AND P1.manufacturer = cname
AND P2.manufacturer = cname
```
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 a_1, a_2, ..., a_k
FROM R_1 AS x_1, R_2 AS x_2, ..., R_n AS x_n
WHERE Conditions
```

```python
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, ..., a_k)\}
return Answer
```
Using the Formal Semantics

What do these queries compute?

\[
\begin{align*}
\text{SELECT DISTINCT } & R.A \\
\text{FROM } & R, S \\
\text{WHERE } & R.A=S.A
\end{align*}
\]

Returns \( R \cap S \)

\[
\begin{align*}
\text{SELECT DISTINCT } & R.A \\
\text{FROM } & R, S, T \\
\text{WHERE } & R.A=S.A \text{ OR } R.A=T.A
\end{align*}
\]

Returns \( R \cap (S \cup T) \) if \( S \neq \emptyset \) and \( T \neq \emptyset \)
Joins Introduce Duplicates

Product \((\text{pname}, \text{price}, \text{category}, \text{manufacturer})\)
Company \((\text{cname}, \text{stockPrice}, \text{country})\)

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

\[
\begin{array}{ll}
\text{SELECT} & \text{Country} \\
\text{FROM} & \text{Product, Company} \\
\text{WHERE} & \text{Manufacturer} = \text{CName} \text{ AND Category} = \text{‘Gadgets’}
\end{array}
\]
## 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 Query

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

**Duplicates! Remember to add DISTINCT**
Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - 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, cid)
Company(cid, cname, city)

For each product return the city where it is manufactured

\[
\text{SELECT X.pname, (SELECT Y.city FROM Company Y WHERE Y.cid=X.cid)} \]

\[
\text{FROM Product X}
\]

What happens if the subquery returns more than one city?
We get a runtime error
1. Subqueries in SELECT

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

Whenever possible, don’t use a nested queries:

```
SELECT pname, (SELECT city FROM Company WHERE Company.cid=Product.cid)
FROM Product
```

```
SELECT pname, city
FROM Product, Company
WHERE Product.cid=Company.cid
```

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

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

Compute the number of products made in by each company

```
SELECT DISTINCT C.cname, (SELECT count(*)
FROM Product P
WHERE P.cid=C.cid)
FROM Company C
```

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

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

Find all products whose prices is > 20 and < 30

```
SELECT P.name
FROM (SELECT * FROM Product WHERE price > 20) as P
WHERE P.price < 30
```

Unnest this query!
3. Subqueries in WHERE

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

Find all companies that make some products with price < 100

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT *
    FROM Product P
    WHERE C.cid = P.cid and P.price < 100)
```
3. Subqueries in WHERE

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

Find all companies that make some products with price < 100

Using **IN**

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price < 100)
```
3. Subqueries in WHERE

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

Find all companies that make some products with price < 100

Using **ANY**:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 100 > ANY (SELECT price
    FROM Product P
    WHERE P.cid = C.cid)
```
3. Subqueries in WHERE

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

Existential quantifiers

Find all companies that make some products with price < 100

Now let’s unnest it:

```
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid= P.cid and P.price < 100
```

Existential quantifiers are easy ! 😊
3. Subqueries in WHERE

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

Find all companies that make only products with price < 100

same as:
Find all companies whose products all have price < 100

Universal quantifiers are hard!
3. Subqueries in WHERE

1. Find the other companies: i.e. s.t. some product ≥ 100

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
FROM Product P
WHERE P.price >= 100)
```

2. Find all companies s.t. all their products have price < 100

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

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

Find all companies that make only products with price < 100

Using EXISTS:

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

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

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

Using **ALL**:

```
SELECT DISTINCT C.cname
FROM   Company C
WHERE  100 > ALL (SELECT price
                  FROM   Product P
                  WHERE  P.cid = C.cid)
```
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**: Query with 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.

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