

Lecture 02: SQL

Wednesday, March 28, 2007

Administrivia

- Homework 1 is out. Due: Fri., April 6
- Did you login on IPROJSRV ?
- Did you change your password ?
- Did you subscribe to CSE444 ?
- Did you read today's reading assignment ?

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

Table name

Attribute names

Tables in SQL

Product

PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

Tuples or rows

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)

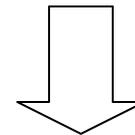
```
SELECT <attributes>  
FROM   <one or more relations>  
WHERE  <conditions>
```

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

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



“selection”

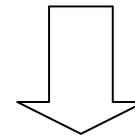
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

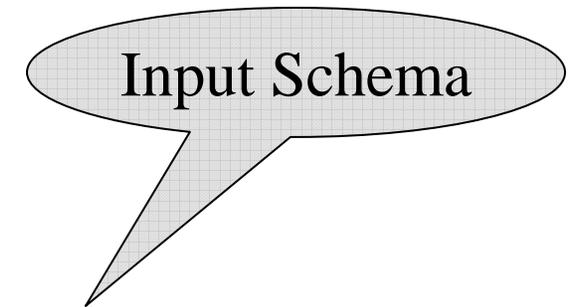
```
SELECT PName, Price, Manufacturer  
FROM Product  
WHERE Price > 100
```



“selection” and
“projection”

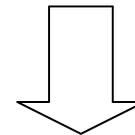
PName	Price	Manufacturer
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

Notation

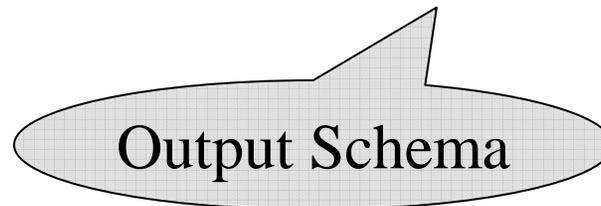


Product(PName, Price, Category, Manufacturer)

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



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

```
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

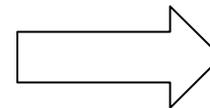
```
SELECT DISTINCT category  
FROM Product
```



Category
Gadgets
Photography
Household

Compare to:

```
SELECT category  
FROM Product
```



Category
Gadgets
Gadgets
Photography
Household

Ordering the Results

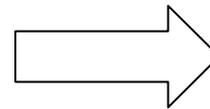
```
SELECT pname, price, manufacturer  
FROM Product  
WHERE category='gizmo' AND price > 50  
ORDER BY price, pname
```

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

Ordering is ascending, unless you specify the DESC keyword.

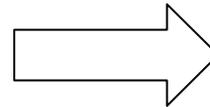
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

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



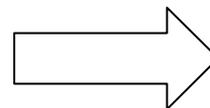
?

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



?

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



?

Keys and Foreign Keys

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key

Product

<u>PName</u>	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

Foreign
key

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
```

Join
between Product
and Company

Joins

Product

PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

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



PName	Price
SingleTouch	\$149.99

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

Product

<u>Name</u>	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

Company

<u>Cname</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

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



Country
??
??

What is the problem ?

A Subtlety about Joins

Product

<u>Name</u>	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

Company

<u>Cname</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

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



Country
USA
USA

Duplicates !
What's the
solution ?

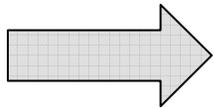
Tuple Variables

Person(pname, address, worksfor)

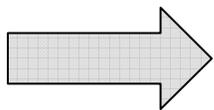
Company(cname, address)

```
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
```

Meaning (Semantics) of SQL Queries

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

```
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

```
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 $S = \phi$?

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

```
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');
```

```
SELECT DISTINCT Company.city
FROM Company, Product, Purchase
WHERE Company.name= Product.maker
AND Product.pname = Purchase.product
AND Purchase.buyer = 'Joe Blow'
```

Now
they are
equivalent

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”

```
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

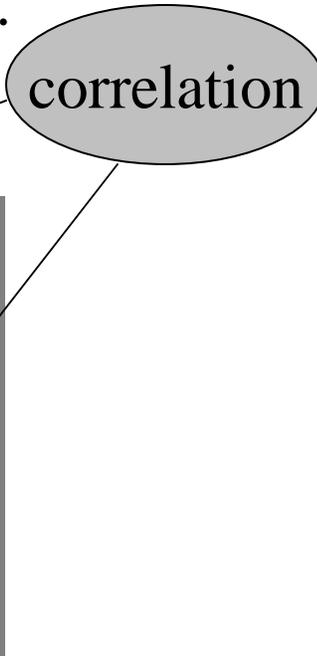
Correlated Queries

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);
```

correlation



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.