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 !

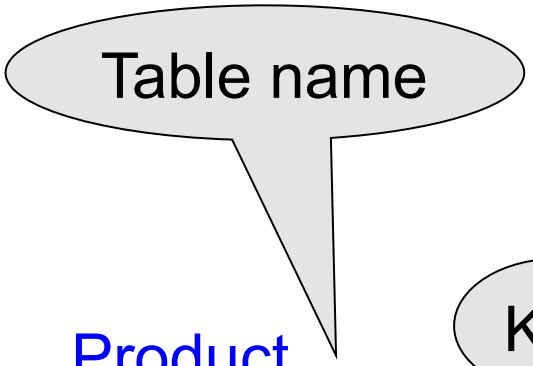
Outline

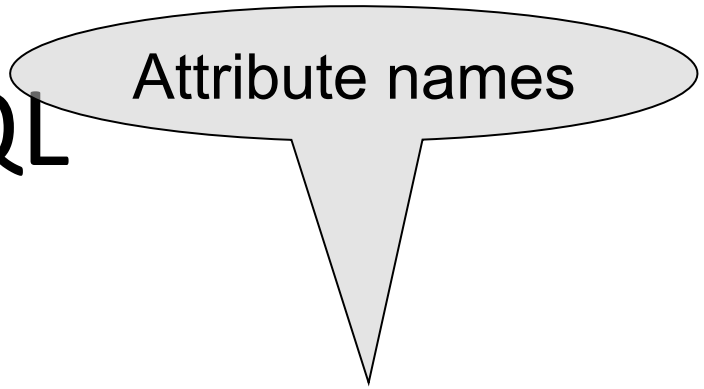
- Data in SQL
- Simple Queries in SQL (6.1)
- Queries with more than one relation (6.2)
- Subqueries (6.3)

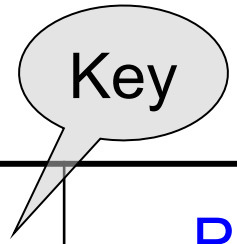
SQL

- 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

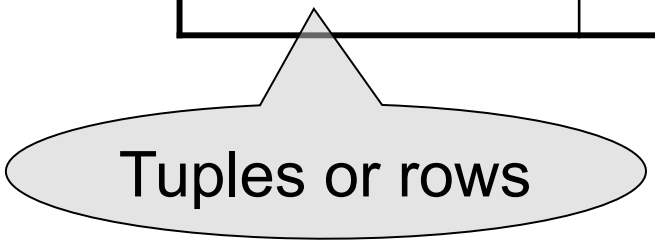






Product

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



Data Types in SQL

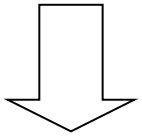
- Atomic types:
  - Characters: CHAR(20), VARCHAR(50)
  - Numbers: INT, BIGINT, SMALLINT, FLOAT
  - Others: MONEY, DATETIME, …
- Record (aka tuple)
  - Has atomic attributes
- Table (relation)
  - A set of tuples

Simple SQL Query

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'
```



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

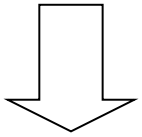
"selection"

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

Details

- Case insensitive:

  SELECT = Select = select

  Product = product

  BUT: 'Seattle' ≠ 'seattle'

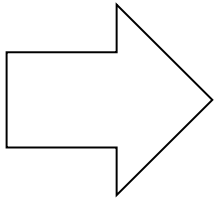
- Constants:

  'abc' - yes

  "abc" - no

Eliminating Duplicates

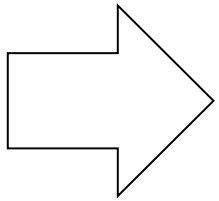
```
SELECT  DISTINCT category
FROM    Product
```



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.

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

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

Key

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

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

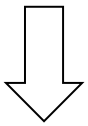
Product

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
```

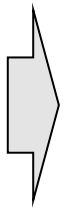


PName	Price
SingleTouch	$149.99

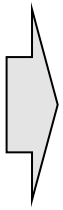
Tuple Variables

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
```

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 $a_1, a_2, \ldots, a_k$
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 ∪ $\{(a_1,\ldots,a_k)\}$
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 $S \neq \varnothing$ and $T \neq \varnothing$
then returns $R \cap (S \cup T)$
else returns $\varnothing$

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

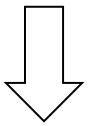
Product

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



Country
USA
USA

Duplicates !
Remember to add DISTINCT

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:

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

=

```
SELECT pname, city
FROM   Product, Company
WHERE cname=company
```

We have "unnested" the query

1. Subqueries in SELECT

Product (pname, price, company)

Company(cname, city)

Compute the number of products made in each city

```
SELECT DISTINCT city, (SELECT count(*)
                       FROM Product
                       WHERE cname=company)
FROM  Company
```

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)

Existential quantifiers

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)

Existential quantifiers

Find all cities that make some products with price < 100

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

$$\{ y \mid \exists x.\ \text{Company}(x,y) \wedge (\exists z.\ \exists p.\ \text{Product}(z,p,x) \wedge p < 100) \}$$

3. Subqueries in WHERE

Product (pname, price, company)

Company(cname, city)

Existential quantifiers

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 Produc.price < 100)
```

3. Subqueries in WHERE

Product (pname, price, company)

Company(cname, city)

Existential quantifiers

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 (pname, price, company)

Company(cname, city)

Existential quantifiers

Find all cities that make some products with price < 100

Now let's unnest it:

```
SELECT DISTINCT  Company.cname
FROM     Company, Product
WHERE  Company.cname = Product.company and Product.price < 100
```

Existential quantifiers are easy ! ☺

3. Subqueries in WHERE

Product (pname, price, company)

Company(cname, city)

Universal quantifiers

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)

Universal quantifiers

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

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

$$\{ y \mid \exists x.\ Company(x,y) \wedge (\forall z.\ \forall p.\ Product(z,p,x) \rightarrow p < 100) \}$$

3. Subqueries in WHERE

De Morgan's Laws:

$$\neg(A \wedge B) = \neg A \vee \neg B$$
$$\neg(A \vee B) = \neg A \wedge \neg B$$
$$\neg \forall x.\ P(x) = \exists x.\ \neg P(x)$$
$$\neg \exists x.\ P(x) = \forall x.\ \neg P(x)$$

$$\neg(A \rightarrow B) = A \wedge \neg B$$

$$\{ y \mid \exists x.\ Company(x,y) \wedge (\forall z.\ \forall p.\ Product(z,p,x) \rightarrow p < 100) \}$$

=

$$\{ y \mid \exists x.\ Company(x,y) \wedge \neg (\exists z \exists p.\ Product(z,p,x) \wedge p \geq 100) \}$$

=

$$\{ y \mid \exists x.\ Company(x,y)) \} \ -$$
$$\{ y \mid \exists x.\ Company(x,y) \wedge (\exists z \exists p.\ Product(z,p,x) \wedge p \geq 100) \}$$

3. Subqueries in WHERE

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

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

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

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE  Company.cname NOT IN (SELECT Product.company
                             FROM Product
                             WHERE Produc.price >= 100
```

3. Subqueries in WHERE

Product (pname, price, company)

Company(cname, city)

Universal quantifiers

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 Produc.price >= 100)
```

3. Subqueries in WHERE

Product (pname, price, company)

Company(cname, city)

Universal quantifiers

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: $\exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$

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

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$

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

x: $\exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

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

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