

Introduction to Database Systems

CSE 444

Lecture 2: SQL

Announcements

- ▶ **Project 1 & Hw 1 are posted on class website**
 - ▶ Project 1 (SQL) due in two weeks
 - ▶ Homework 1 (E/R models etc) due in three weeks
 - ▶ Remember: time goes by very fast! Start early!

- ▶ **On the course website you will find**
 - ▶ Recommended readings from the book
 - ▶ PDF of lecture notes (~morning of class)

- ▶ **Other**

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

Attribute names

Table name

Product

Key

<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

Tuple / row

Attribute

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 (aka relation)
 - ▶ A set of tuples

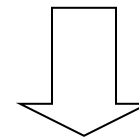
Book Sec. 2.3.2

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

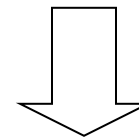
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks

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
& Projection**

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

Details

- ▶ SQL is case insensitive
 - ▶ SELECT = Select = select
 - ▶ Product = product
 - ▶ but 'Seattle' ≠ 'seattle' (in general)

- ▶ Constants must use single quotes
 - ▶ 'abc' - yes
 - ▶ "abc" - no

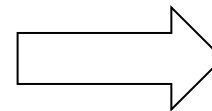
Eliminating Duplicates

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

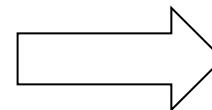
Set vs. Bag semantics

```
SELECT category  
FROM Product
```



Category
Gadgets
Gadgets
Photography
Household

```
SELECT DISTINCT category  
FROM Product
```



Category
Gadgets
Photography
Household

Ordering the Results

```
SELECT pName, price, manufacturer
FROM Product
WHERE category='Gadgets'
      and price > 10
ORDER BY price, pName
```

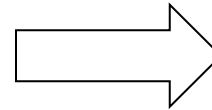
- ▶ Ties in price *attribute* broken by *pname* attribute
- ▶ Ordering is ascending by default. Descending:

```
... ORDER BY price, pname DESC
```

Product

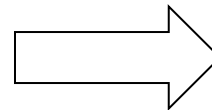
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

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



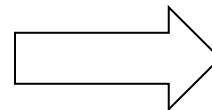
?

```
SELECT category  
FROM Product  
ORDER BY pName
```



?

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

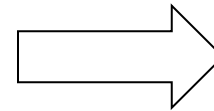


?

Product

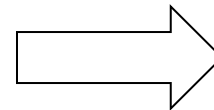
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

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



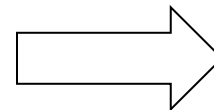
Category
Gadgets
Household
Photography

```
SELECT category  
FROM Product  
ORDER BY pName
```



Category
Gadgets
Household
Gadgets
Photography

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



Syntax error*

* Error actually happens during "semantic" analysis of query.

Keys and Foreign Keys

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

Key → Foreign key

Company

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

Key →

Joins

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

*Q: 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 b/w
Product and
Company



Joins

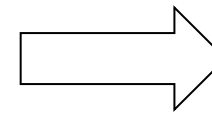
Product

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

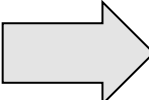
Tuple Variables

Person (pName, address, works_for)
Company (cName, address)

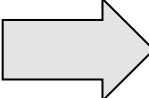
```
SELECT DISTINCT pName, address
FROM Person, Company
WHERE works_for = cName
```



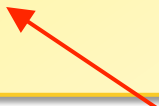
which address?



```
SELECT DISTINCT Person.pName, Company.address
FROM Person, Company
WHERE Person.works_for = Company.cName
```



```
SELECT DISTINCT X.pName, Y.address
FROM Person as X, Company as Y
WHERE X.works_for = Y.cName
```



"as" is optional

In Class

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: 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)

Q: Find all Chinese companies that manufacture products in the 'Toy' category!

```
SELECT DISTINCT cName
FROM Product P, Company
WHERE country = 'China'
      and P.category = 'Toy'
      and P.manufacturer = cName
```

In Class

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all Chinese companies that manufacture products both in the 'Toy' and 'Electronic' categories.

```
SELECT DISTINCT cName  
FROM  
WHERE
```

In Class

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all Chinese companies that manufacture products both in the 'Toy' and 'Electronic' 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
```

Meaning (Semantics) of SQL Queries

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

Conceptual evaluation strategy (nested for loops):

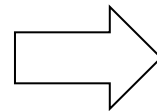
```
Answer = {}  
for x1 in R1 do  
  for x2 in R2 do  
    .....  
    for xn in Rn do  
      if Conditions  
        then Answer = Answer ∪ {(a1, ..., ak)}  
return Answer
```

Using the Formal Semantics

What do these queries compute?

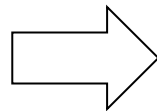
$R(a), S(a), T(a)$

```
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)

Q: Find all Chinese companies that manufacture some product in the 'Gadgets' category!

```
SELECT country
FROM Product, Company
WHERE manufacturer = cName
      and category = 'Gadgets'
```


Joins Introduce Duplicates

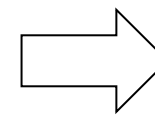
Product

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 country
FROM Product, Company
WHERE manufacturer = cName
and category = 'Gadgets'
```



Country
USA
USA

Remember to use **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)

Q: For each product return the city where it is manufactured!

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

What happens if the subquery returns more than one city ?

Runtime error

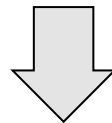
1. Subqueries in SELECT

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

Q: For each product return the city where it is manufactured!

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

"unnesting the query"



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

Whenever possible,
don't use nested queries

1. Subqueries in SELECT

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

Q: Compute the number of products made by each company!

```
SELECT 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)

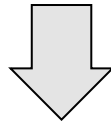
2. Subqueries in FROM

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

Q: Find all products whose prices is > 20 and < 30!

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

unnesting



```
SELECT pname
FROM Product
WHERE price > 20 and price < 30
```

3. Subqueries in WHERE

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

Existential quantifiers \exists

Q: 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)

Existential quantifiers \exists

Q: 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)

Existential quantifiers \exists

Q: 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 \exists

Q: Find all companies that make some products with price < 100!

Now, let's unnest:

```
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)

Universal quantifiers \forall

Q: Find all companies that make only products with price < 100!

same as:

Q: Find all companies for which all products have price < 100!

Universal quantifiers are more complicated ! ☹️

3. Subqueries in WHERE

1. Find the other companies: i.e. they have **some** product ≥ 100 !

```
SELECT DISTINCT C.cname
FROM   Company C
WHERE  C.cid IN ( SELECT P.cid
                  FROM   Product P
                  WHERE  P.price  $\geq$  100)
```

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

```
SELECT DISTINCT C.cname
FROM   Company C
WHERE  C.cid NOT IN ( SELECT P.cid
                     FROM   Product P
                     WHERE  P.price  $\geq$  100)
```

3. Subqueries in WHERE

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

Universal quantifiers \forall

Q: Find all companies that make only products with price < 100!

Using **NOT EXISTS**:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT 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)

Universal quantifiers \forall

Q: 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 & Friends

- ▶ How can we unnest the *universal quantifier* query ?

Queries that must be nested

- ▶ 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**
 - ▶ Add one tuple violating the condition. Then not "all"...
- ▶ **Consequence: we cannot unnest a query with a universal quantifier**
- ▶ **Same argument holds for queries with negation**

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