

SQL

# Outline

- *Data* in SQL
- Simple Queries in SQL
- Queries with more than one relation

# 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 of these
- What we discuss is common to all of them

# 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
- Transact-SQL
  - Idea: package a sequence of SQL statements → server
  - Won't discuss in class

# Data in SQL

1. Atomic types, a.k.a. data types
2. Tables built from atomic types

Unlike XML, no nested tables, only flat tables are allowed!

- We will see later how to decompose complex structures into multiple flat tables

# Data Types in SQL

- Characters:
  - CHAR(20) -- fixed length
  - VARCHAR(40) -- variable length
- Numbers:
  - BIGINT, INT, SMALLINT, TINYINT
  - REAL, FLOAT -- differ in precision
  - MONEY
- Times and dates:
  - DATE
  - DATETIME -- SQL Server
- Others... All are simple

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

- 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()**.



# 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)

# SQL Query

Basic form: (plus many many more bells and whistles)

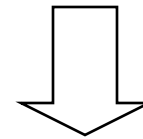
<b>SELECT</b>	attributes
<b>FROM</b>	relations (possibly multiple)
<b>WHERE</b>	conditions (selections)

# 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

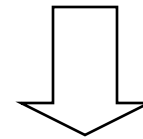
“selection”

# 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”

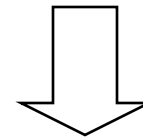
PName	Price	Manufacturer
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

# A Notation for SQL Queries

Input Schema

Product(PName, Price, Category, Manufacturer)

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



Answer(PName, Price, Manufacturer)

Output Schema

# Selections

What goes in the **WHERE** clause:

- $x = y$ ,  $x < y$ ,  $x \leq y$ , etc
  - For number, they have the usual meanings
  - For CHAR and VARCHAR: lexicographic ordering
    - Expected conversion between CHAR and VARCHAR
  - For dates and times, what you expect...
- Pattern matching on strings...

# The **LIKE** operator

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - \_ = any single character

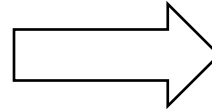
Product(PName, Price, Category, Manufacturer)

Find all products whose name mentions 'gizmo':

```
SELECT *  
FROM Products  
WHERE PName LIKE '%gizmo%'
```

# Eliminating Duplicates

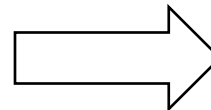
```
SELECT DISTINCT category  
FROM Product
```



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
```

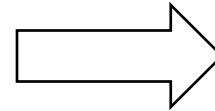
Ordering is ascending, unless you specify the DESC keyword.

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

# Ordering the Results

```
SELECT category  
FROM Product  
ORDER BY pname
```

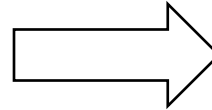
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi



?

# Ordering the Results

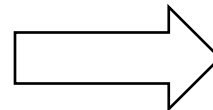
```
SELECT DISTINCT category  
FROM Product  
ORDER BY category
```



Category
Gadgets
Household
Photography

Compare to:

```
SELECT category  
FROM Product  
ORDER BY pname
```



?

# Joins in SQL

- Connect two or more tables:

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

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

What is  
the connection  
between  
them ?

# 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 in SQL

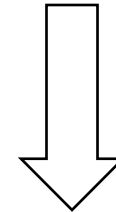
Product

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

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

# 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'
```

# Joins in SQL

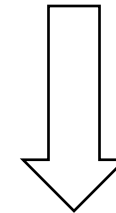
Product

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
??
??

What is  
the problem ?  
What's the  
solution ?



# Joins

Product (pname, price, category, manufacturer)

Purchase (buyer, seller, store, product)

Person(persname, phoneNumber, city)

Find names of people living in Seattle that bought some product in the ‘Gadgets’ category, and the names of the stores they bought such product from

```
SELECT DISTINCT persname, store
FROM   Person, Purchase, Product
WHERE  persname=buyer AND product = pname AND
       city='Seattle' AND category='Gadgets'
```

# When are two tables related?

- You guess they are
- I tell you so
- Foreign keys are a method for schema designers to tell you so (7.1)
  - A foreign key states that a column is a reference to the key of another table  
ex: **Product.manufacturer** is foreign key of **Company**
  - Gives information and enforces constraint

# Disambiguating Attributes

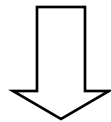
- Sometimes two relations have the same attr:

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
```

# Tuple Variables

Product (pname, price, category, manufacturer)

Purchase (buyer, seller, store, product)

Person(persname, phoneNumber, city)

Find all stores that sold at least one product that the store 'BestBuy' also sold:

```
SELECT DISTINCT x.store
FROM   Purchase AS x, Purchase AS y
WHERE  x.product = y.product AND y.store = 'BestBuy'
```

Answer (store)

# Tuple Variables

General rule:

tuple variables introduced automatically by the system:

Product (name, price, category, manufacturer)

```
SELECT name  
FROM   Product  
WHERE  price > 100
```

Becomes:

```
SELECT Product.name  
FROM   Product AS Product  
WHERE  Product.price > 100
```

Doesn't work when Product occurs more than once:  
In that case the user needs to define variables explicitly.

# Meaning (Semantics) of SQL Queries

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

## 1. Nested loops:

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

# Meaning (Semantics) of SQL Queries

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

## 2. Parallel assignment

```
Answer = { }  
for all assignments x1 in R1, ..., xn in Rn do  
    if Conditions then Answer = Answer  $\cup$  {(a1,...,ak)}  
return Answer
```

Doesn't impose any order !

# First Unintuitive SQLism

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

Looking for  $R \cap (S \cup T)$

But what happens if T is empty?



# Exercises

Product (pname, price, category, manufacturer)

Purchase (buyer, seller, store, product)

Company (cname, stock price, country)

Person(per-name, phone number, city)

Ex #1: Find people who bought telephony products.

Ex #2: Find names of people who bought American products

Ex #3: Find names of people who bought American products and they live in Seattle.

Ex #4: Find people who have both bought and sold something.

Ex #5: Find people who bought stuff from Joe or bought products from a company whose stock prices is more than \$50.