Principles of Database Systems
CSE 544p

Lecture #1
September 29, 2010
Staff

• Instructor: Dan Suciu
  – CSE 662, suciu@cs.washington.edu
  Office hours: by appointment

• TAs:
  – Jessica Leung joyleung@cs.washington.edu
  – TA: Paramjit Singh Sandhu paramsan@cs.washington.edu

• Technical support:
  – At UW: Fred Videon, fred@cs.washington.edu
  – At MS: Matt McGinley
Communications

• Web page:
  http://www.cs.washington.edu/p544
  – Lectures will be available here
  – Homework will be posted here
  – Announcements may be posted here

• Mailing list:
  – Announcements, group discussions
  – Please subscribe
Textbook(s)

Main textbook:
• *Database Management Systems*, Ramakrishnan and Gehrke

Second textbook:
Course Format

• Lectures Wednesdays, 6:30-9:20

• 7 Homework Assignments

• Take-home Final
Grading

• Homework: 70%

• Take-home Final: 30%
Homework Assignments

1. SQL
2. Conceptual design
3. JAVA/SQL
4. Transactions
5. Database tuning
6. XML/XPath/XQuery
7. Pig Latin, on AWS

Due: Tuesdays’, by 11:59pm. Three late days per person
Take-home Final

• Posted on December 8, at 11:59pm

• Due on December 9, by 11:59pm

• No late days/hours/minutes/seconds

December 9th is the day of your final
Software Tools

• SQL Server 2008
  – You have access to http://msdnaa.cs.washington.edu
  – Username is full @cs.washington.edu email address
  – Doesn’t work? Email ms-sw-admin@cs.washington.edu
  – Download the client, connect to IPROJSRV (may need tunneling)
  – OK to use your own server, just import IMDB (may need tunneling)

• Postgres: download from
  – download http://www.postgresql.org/download/
  – Is also installed on lab machines

• Xquery: download one interpreter from
  – Zorba: http://www.zorba-xquery.com/ (I use this one: ½ day installation)
  – Galax: http://galax.sourceforge.net/ (great in the past, seems less well maintained)
  – Saxon: http://saxon.sourceforge.net/ (from apache; very popular)

• Pig Latin: download from
  – We will also run it on Amazon Web Services
Rest of Today’s Lecture

• Overview of DBMS

• SQL
Database

What is a database?

Give examples of databases
Database

What is a database?
• A collection of files storing related data

Give examples of databases
• Accounts database; payroll database; UW’s students database; Amazon’s products database; airline reservation database
Database Management System

What is a DBMS?

Give examples of DBMS
Database Management System

What is a DBMS?

• A big C program written by someone else that allows us to manage efficiently a large database and allows it to persist over long periods of time

Give examples of DBMS

• DB2 (IBM), SQL Server (MS), Oracle, Sybase
• MySQL, Postgres, ...

SQL for Nerds, Greenspun, http://philip.greenspun.com/sql/ (Chap 1,2)
Market Shares

From 2006 Gartner report:

- IBM: 21% market with $3.2BN in sales
- Oracle: 47% market with $7.1BN in sales
- Microsoft: 17% market with $2.6BN in sales
An Example

The Internet Movie Database
http://www.imdb.com

• Entities:
  Actors (800k), Movies (400k), Directors, ...

• Relationships:
  who played where, who directed what, ...
# Tables

## Actor:

<table>
<thead>
<tr>
<th>id</th>
<th>fName</th>
<th>lName</th>
<th>gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>195428</td>
<td>Tom</td>
<td>Hanks</td>
<td>M</td>
</tr>
<tr>
<td>645947</td>
<td>Amy</td>
<td>Hanks</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

## Cast:

<table>
<thead>
<tr>
<th>pid</th>
<th>mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>195428</td>
<td>337166</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

## Movie:

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>337166</td>
<td>Toy Story</td>
<td>1995</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SQL

```sql
SELECT *
FROM Actor
```
SQL

SELECT count(*)
FROM Actor

This is an aggregate query
SQL

SELECT *
FROM Actor
WHERE lName = 'Hanks'

This is a selection query
SQL

SELECT * 
FROM Actor, Casts, Movie 
WHERE lname='Hanks' and Actor.id = Casts.pid 
    and Casts.mid=Movie.id and Movie.year=1995

This query has selections and joins

817k actors, 3.5M casts, 380k movies; 
How can it be so fast?
How Can We Evaluate the Query?

<table>
<thead>
<tr>
<th>Actor:</th>
<th>Cast:</th>
<th>Movie:</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>fName</td>
<td>lName</td>
</tr>
<tr>
<td>pid</td>
<td>mid</td>
<td></td>
</tr>
</tbody>
</table>

Plan 1: . . . . [ in class ]

Plan 2: . . . . [ in class ]
Evaluating Tom Hanks

\[ \sigma_{\text{Name}=\text{Hanks}} \]

\[ \sigma_{\text{year}=1995} \]

\[ \sigma_{\text{Name}=\text{Hanks}} \]

\[ \sigma_{\text{year}=1995} \]
Optimization and Query Execution

• Indexes: on Actor.lName, on Movie.year

• Query optimization
  – Access path selection
  – Join order

• Statistics

• Multiple implementations of joins
Recovery

• Transfer $100 from account #4662 to #7199:

\[
\begin{align*}
X &= \text{Read(Account\_1);} \\
X.\text{amount} &= X.\text{amount} - 100; \\
\text{Write(Account\_1, X);} \\
Y &= \text{Read(Account\_2);} \\
Y.\text{amount} &= Y.\text{amount} + 100; \\
\text{Write(Account\_2, Y);} \\
\end{align*}
\]
Recovery

• Transfer $100 from account #4662 to #7199:

X = Read(Account_1);
X.amount = X.amount - 100;
Write(Account_1, X);

Y = Read(Account_2);
Y.amount = Y.amount + 100;
Write(Account_2, Y);

What is the problem?
Concurrent Control

- How to overdraft your account:

User 1

```c
X = Read(Account);
if (X.amount > 100)
    { dispense_money( );
        X.amount = X.amount - 100;
    }
else error("Insufficient funds");
```

User 2

```c
X = Read(Account);
if (X.amount > 100)
    { dispense_money( );
        X.amount = X.amount - 100;
    }
else error("Insufficient funds");
```

What can go wrong?
Transactions

• Recovery

• Concurrency control

ACID =
• Atomicity ( = recovery)
• Consistency
• Isolation ( = concurrency control)
• Durability
Client/Server Database Architecture

• There is one single server that stores the database (called DBMS or RDBMS):
  – Usually a beefed-up system, e.g. IPROJSRV
  – But can be your own desktop...
  – ... or a huge cluster running a parallel dbms
• Many clients running apps and connecting to DBMS
  – E.g. Microsoft’s Management Studio
  – Or psql (for postgres)
  – Always: some else’s big Java or C++ program
• The client “talks” to the server using JDBC protocol
Types of Usages for Databases

• OLTP (online-transaction-processing)
  – Many updates
  – Many “point queries”: retrieve the record with a given key.

• Decision-Support
  – Many aggregate/group-by queries.
  – Sometimes called *data warehouse*
SQL v.s. noSQL

• Reading for next time:
Data Management

• Data Management is more than databases!

Here is an example of a problem:
• Alice sends Bob in random order all the numbers 1, 2, 3, ..., 100000000000000000000
• She does not repeat any number
• But she misses exactly one
• Help Bob find out which one is missing!

After you solve it, make it a bit harder:
• Alice misses exactly ten numbers
• Help Bob find out which ones are missing!
Accessing SQL Server

• SQL Server Management Studio
• Server Type = Database Engine
• Server Name = IPROJSRV
• Authentication = SQL Server Authentication
  – Login = your UW email address (not the CSE email)
  – Password = [in class]
• Must connect from within CSE, or must use tunneling
• Alternatively: install your own, get it from MSDNAA (see earlier slide)
• Then play with IMDB, start working on HW 1
Outline for Today

• Basics: we go quickly or skip slides, please read the slides at home
  – Datatypes in SQL
  – Simple Queries in SQL
  – Joins

• Subqueries: this is tough! Please read the relational calculus and tuple calculus in the textbook (Chapter 4.3)

• Aggregates: separates pros from amateurs

• Nulls, Outer joins
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

<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>

**Table name**

**Attribute names**

**Tuples or rows**

**Key**
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

<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 * FROM Product WHERE category='Gadgets'

“selection”
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

• Case insensitive:

  \[ \text{SELECT} = \text{Select} = \text{select} \]
  \[ \text{Product} = \text{product} \]
  \[ \text{BUT: ‘Seattle’} \neq \text{‘seattle’} \]

• Constants:

  ‘abc’ - yes
  “abc” - no
Eliminating Duplicates

```
SELECT DISTINCT category
FROM Product
```

Compare to:

```
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>
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.
### SQL Queries

1. **Selecting Distinct Categories**

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

2. **Selecting Category and PName**

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

3. **Selecting Distinct Categories again**

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

---

### Table

<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>
# 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

Dan Suciu -- p544 Fall 2010
Tuple Variables

Person\( (\text{pname}, \text{address}, \text{worksfor}) \)

Company\( (\text{cname}, \text{address}) \)

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT} \quad \text{pname, address} \\
\text{FROM} & \quad \text{Person, Company} \\
\text{WHERE} & \quad \text{worksfor} = \text{cname}
\end{align*}
\]

Which address?

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT} \quad \text{Person.pname, Company.address} \\
\text{FROM} & \quad \text{Person, Company} \\
\text{WHERE} & \quad \text{Person.worksfor} = \text{Company.cname}
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT} \quad x.pname, y.address \\
\text{FROM} & \quad \text{Person AS } x, \text{ Company AS } y \\
\text{WHERE} & \quad x.worksfor = y.cname
\end{align*}
\]
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    
```
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, ..., 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, ..., a_k)\}
return Answer
```
Using the Formal Semantics

What do these queries compute?

\[
\text{SELECT DISTINCT R.A FROM R, S WHERE R.A=S.A}
\]

Returns \( R \cap S \)

\[
\text{SELECT DISTINCT R.A FROM R, S, T WHERE R.A=S.A OR R.A=T.A}
\]

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)

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

<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

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

### Duplicates!

Remember to add `$DISTINCT$` to avoid duplicates.

---

Dan Suciu -- p544 Fall 2010
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)

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 Product.price < 100)
```
3. Subqueries in WHERE

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

Find all cities that make some products with price < 100

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

\{ y | \exists x. \text{Company}(x,y) \land (\exists z. \exists p. \text{Product}(z,p,x) \land p < 100) \}
3. Subqueries in WHERE

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

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 Product.price < 100)
```
3. Subqueries in WHERE

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

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 (\texttt{pname}, price, company)  
Company(\texttt{cname}, city)

Find all cities that make \texttt{some} products with price < 100

Now let’s unnest it:

\begin{verbatim}
SELECT DISTINCT Company.cname  
FROM Company, Product  
WHERE Company.cname = Product.company and Product.price < 100
\end{verbatim}

Existential quantifiers are easy! 😊
3. Subqueries in WHERE

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

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)

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

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

\{ y | \exists x. \text{Company}(x,y) \land (\forall z. \forall p. \text{Product}(z,p,x) \implies p < 100) \}
3. Subqueries in WHERE

De Morgan’s Laws:

\[ \neg(A \land B) = \neg A \lor \neg B \]
\[ \neg(A \lor B) = \neg A \land \neg B \]
\[ \neg \forall x. P(x) = \exists x. \neg P(x) \]
\[ \neg \exists x. P(x) = \forall x. \neg P(x) \]

\{ y \mid \exists x. \text{Company}(x,y) \land (\forall z. \forall p. \text{Product}(z,p,x) \Rightarrow p < 100) \} = \\
\{ y \mid \exists x. \text{Company}(x,y) \land \neg (\exists z \exists p. \text{Product}(z,p,x) \land p \geq 100) \} = \\
\{ y \mid \exists x. \text{Company}(x,y) \} \setminus \\
\{ y \mid \exists x. \text{Company}(x,y) \land (\exists z \exists p. \text{Product}(z,p,x) \land p \geq 100) \}
3. Subqueries in WHERE

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

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

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

```sql
SELECT DISTINCT Company.city
FROM Company
WHERE Company.cname NOT IN (SELECT Product.company
                             FROM Product
                             WHERE Produto.price >= 100)
```
3. Subqueries in WHERE

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

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)

Find all cities with companies that make only 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

**Rule of Thumb:**
Non-monotone queries cannot be unnested. In particular, queries with a universal quantifier cannot be unnested.
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.

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

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

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

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

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

SQL supports several aggregation operations:

- sum, count, min, max, avg

Except count, all aggregations apply to a single attribute
Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT Count(category) FROM Product WHERE year > 1995
```

same as Count(*)

We probably want:

```
SELECT Count(DISTINCT category) FROM Product WHERE year > 1995
```
More Examples

Purchase(product, date, price, quantity)

\[
\text{SELECT} \quad \text{Sum(price} \times \text{quantity)} \\
\text{FROM} \quad \text{Purchase}
\]

\[
\text{SELECT} \quad \text{Sum(price} \times \text{quantity)} \\
\text{FROM} \quad \text{Purchase} \\
\text{WHERE} \quad \text{product} = \text{‘bagel’}
\]

What do they mean?

Dan Suciu -- p544 Fall 2010
Simple Aggregations

**Purchase**

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.5</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

**SELECT**  
Sum(price * quantity)  
**FROM**  
Purchase  
**WHERE**  
product = ‘Bagel’

90  (= 60+30)
Grouping and Aggregation

$\text{Purchase(product, price, quantity)}$

Find total quantities for all sales over $1$, by product.

\[
\begin{align*}
\text{SELECT} & \quad \text{product, Sum(quantity) AS TotalSales} \\
\text{FROM} & \quad \text{Purchase} \\
\text{WHERE} & \quad \text{price} > 1 \\
\text{GROUP BY} & \quad \text{product}
\end{align*}
\]

Let’s see what this means…
Grouping and Aggregation

1. Compute the **FROM** and **WHERE** clauses.

2. Group by the attributes in the **GROUPBY**

3. Compute the **SELECT** clause, including aggregates.
1&2. FROM-WHERE-GROUPBY

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.5</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>
3. SELECT

```
SELECT product, Sum(quantity) AS TotalSales
FROM Purchase
WHERE price > 1
GROUP BY product
```
GROUP BY v.s. Nested Queries

```
SELECT product, Sum(quantity) AS TotalSales
FROM Purchase
WHERE price > 1
GROUP BY product
```

```
SELECT DISTINCT x.product, (SELECT Sum(y.quantity) FROM Purchase y WHERE x.product = y.product AND price > 1) AS TotalSales
FROM Purchase x
WHERE price > 1
```

Why twice?
Another Example

```
SELECT product,
       sum(quantity) AS SumSales,
       max(price) AS MaxQuantity
FROM Purchase
GROUP BY product
```

What does it mean?

Rule of thumb:

Every group in a GROUP BY is non-empty!

If we want to include empty groups in the output, then we need either a subquery, or a left outer join (see later)
HAVING Clause

Same query, except that we consider only products that had at least 100 buyers.

```
SELECT  product, Sum(quantity)
FROM    Purchase
WHERE   price > 1
GROUP BY product
HAVING  Sum(quantity) > 30
```

HAVING clause contains conditions on aggregates.
General form of Grouping and Aggregation

```
SELECT S
FROM R_1,...,R_n
WHERE C1
GROUP BY a_1,...,a_k
HAVING C2
```

S = may contain attributes a_1,...,a_k and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in R_1,...,R_n

C2 = is any condition on aggregate expressions

Why?
General form of Grouping and Aggregation

```
SELECT  S
FROM    R_1,...,R_n
WHERE   C_1
GROUP BY a_1,...,a_k
HAVING  C_2
```

Evaluation steps:
1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes a_1,...,a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result
Advanced SQLizing

1. Unnesting Aggregates

2. Finding witnesses
Unnesting Aggregates

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

Find the number of companies in each city

SELECT DISTINCT city, (SELECT count(*)
FROM Company Y
WHERE X.city = Y.city)
FROM Company X

SELECT city, count(*)
FROM Company
GROUP BY city

Equivalent queries

Note: no need for DISTINCT  
(DISTINCT is the same as GROUP BY)
Unnesting Aggregates

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

Find the number of products made in each city

```
SELECT DISTINCT X.city, (SELECT count(*)
FROM Product Y, Company Z
WHERE Y.cname = Z.company
AND Z.city = X.city)
FROM Company X
```

They are NOT equivalent! (WHY?)
More Unnesting

Author(login,name)
Wrote(login,url)

• Find authors who wrote ≥ 10 documents:
• Attempt 1: with nested queries

SELECT DISTINCT Author.name
FROM Author
WHERE count(SELECT Wrote.url
    FROM Wrote
    WHERE Author.login=Wrote.login)
> 10

This is SQL by a novice
More Unnesting

- Find all authors who wrote at least 10 documents:
- Attempt 2: SQL style (with GROUP BY)

```
SELECT Author.name
FROM Author, Wrote
WHERE Author.login=Wrote.login
GROUP BY Author.name
HAVING count(wrote.url) > 10
```

This is SQL by an expert
Finding Witnesses

Store($sid$, $sname$)
Product($pid$, $pname$, $price$, $sid$)

For each store,
find its most expensive products
Finding Witnesses

Finding the maximum price is easy…

```
SELECT Store.sid, max(Product.price)
FROM Store, Product
WHERE Store.sid = Product.sid
GROUP BY Store.sid
```

But we need the *witnesses*, i.e. the products with max price
Finding Witnesses

To find the witnesses, compute the maximum price in a subquery

```
SELECT Store.sname, Product.pname
FROM Store, Product,
    (SELECT Store.sid AS sid, max(Product.price) AS p
     FROM Store, Product
     WHERE Store.sid = Product.sid
     GROUP BY Store.sid, Store.sname) X
WHERE Store.sid = Product.sid
    and Store.sid = X.sid and Product.price = X.p
```
Finding Witnesses

There is a more concise solution here:

\[
\begin{align*}
\text{SELECT} & \quad \text{Store.sname, x.pname} \\
\text{FROM} & \quad \text{Store, Product x} \\
\text{WHERE} & \quad \text{Store.sid = x.sid and} \\
& \quad \text{x.price} \geq \text{ALL (SELECT y.price} \\
& \quad \text{FROM Product y} \\
& \quad \text{WHERE Store.sid = y.sid)}
\end{align*}
\]
NULLS in SQL

• Whenever we don’t have a value, we can put a NULL
• Can mean many things:
  – Value does not exists
  – Value exists but is unknown
  – Value not applicable
  – Etc.
• The schema specifies for each attribute if can be null *(nullable attribute)* or not
• How does SQL cope with tables that have NULLs?
Null Values

• If x= NULL then $4\times(3-x)/7$ is still NULL

• If x= NULL then x='Joe’ is UNKNOWN

• In SQL there are three boolean values:
  
  FALSE = 0
  
  UNKNOWN = 0.5
  
  TRUE = 1
Null Values

- \( C_1 \text{ AND } C_2 = \min(C_1, C_2) \)
- \( C_1 \text{ OR } C_2 = \max(C_1, C_2) \)
- \( \text{NOT } C_1 = 1 - C_1 \)

Rule in SQL: include only tuples that yield TRUE

```sql
SELECT *
FROM Person
WHERE (age < 25) AND (height > 6 OR weight > 190)
```

E.g.

- age=20
- heigth=NULL
- weight=200

Dan Suciu -- p544 Fall 2010
Null Values

Unexpected behavior:

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25
```

Some Persons are not included!
Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM Person
WHERE age < 25 OR age >= 25 OR age IS NULL
```

Now it includes all Persons
Outerjoins

Product(name, category)
Purchase(prodName, store)

An “inner join”:

```
SELECT Product.name, Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
```

Same as:

```
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON
    Product.name = Purchase.prodName
```

But Products that never sold will be lost!
Outerjoins

If we want the never-sold products, need an “outerjoin”:

```sql
SELECT Product.name, Purchase.store
FROM Product LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```
### Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

### Purchase

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
<tr>
<td>OneClick</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Compute, for each product, the total number of sales in ‘September’

Product(name, category)
Purchase(prodName, month, store)

```
SELECT Product.name, count(*)
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
       and  Purchase.month = 'September'
GROUP BY Product.name
```
Application

Compute, for each product, the total number of sales in ‘September’

Product (name, category)
Purchase (prodName, month, store)

```
SELECT Product.name, count(store)
FROM   Product LEFT OUTER JOIN Purchase ON
        Product.name = Purchase.prodName
        and   Purchase.month = 'September'
GROUP BY Product.name
```

Now we also get the products who sold in 0 quantity
Outer Joins

• Left outer join:
  – Include the left tuple even if there’s no match

• Right outer join:
  – Include the right tuple even if there’s no match

• Full outer join:
  – Include the both left and right tuples even if there’s no match