Database Management Systems
CSEP 544

Lecture #1

March 31, 2009

Staff

Instructor: Prof. Dan Suciu:

- Bell Labs, AT&T Labs, UW, MSR
- Research interests:
  - Semi-structured data (XML): XML-QL, XMill (XML compressor), XPath containment
  - Probabilistic Databases
  - Database Security and Privacy
- CSE 662, suciu@cs.washington.edu, Office hours by email appointments (any day OK except Tuesday)

TA: Bhushan Mandhani

Communications

- Web page: http://www.cs.washington.edu/csep544/
  - Lectures are here
  - The homework assignments are here
- Mailing list:
  - Announcements, group discussions
  - Please subscribe

Textbook(s)

Main textbook:

- *Database Management Systems*, Ramakrishnan and Gehrke

Second textbook:

- *Database Systems: The Complete Book*, Garcia-Molina, Ullman, Widom

Most important: COME TO CLASS ! ASK QUESTIONS !

Other Texts

- *Fundamentals of Database Systems*, Elmasri, Navathe
- *XQuery from the Experts*, Katz, Ed.
- *Foundations of Databases*, Abiteboul, Hull, Vianu
- *Data on the Web,* Abiteboul, Buneman, Suciu

Course Format

- Lectures Tuesdays, 6:30-9:20
- 7 Homework Assignments
- Final

Grading

- Homework Assignments: 70%
- Final: 30%

7 Homework Assignments

1. SQL (already posted)
2. Conceptual Design (already posted)
3. SQL in Java
4. Transactions
5. Database tuning
6. Query optimization
7. XQuery

Due: Tuesdays, every week, by email to Bhushan

Final

- Need to reschedule the official date (June 11)

- Proposed date for the final:

  TUESDAY, JUNE 9, 2009, 6:30-8:20pm

- If you can't make it, let me know by email; I'd like to make the date official next week

Outline of Today's Lecture

1. Overview of DBMS

2. Course content

3. 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, …

Market Shares

From 2004 www.computerworld.com

- IMB: 35% market with $2.5BN in sales
- Oracle: 33% market with $2.3BN in sales
- Microsoft: 19% market with $1.3BN 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:

id	**fName**	**lName**	**gender**
195428	Tom	Hanks	M
645947	Amy	Hanks	F
. . .			

Cast:

pid	**mid**
195428	337166
. . .	

Movie:

id	**Name**	**year**
337166	Toy Story	1995
. . .	. . .	. ..

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, Cast, Movie
WHERE lname='Hanks' and Actor.id = Cast.pid
  and Cast.mid=Movie.id and Movie.year=1995
```

This query has *selections* and *joins*

How Can We Evaluate the Query ?

Actor:

id	fName	lName	gender
. . .		Hanks	
. . .			

Cast:

pid	mid
. . .	
. . .	

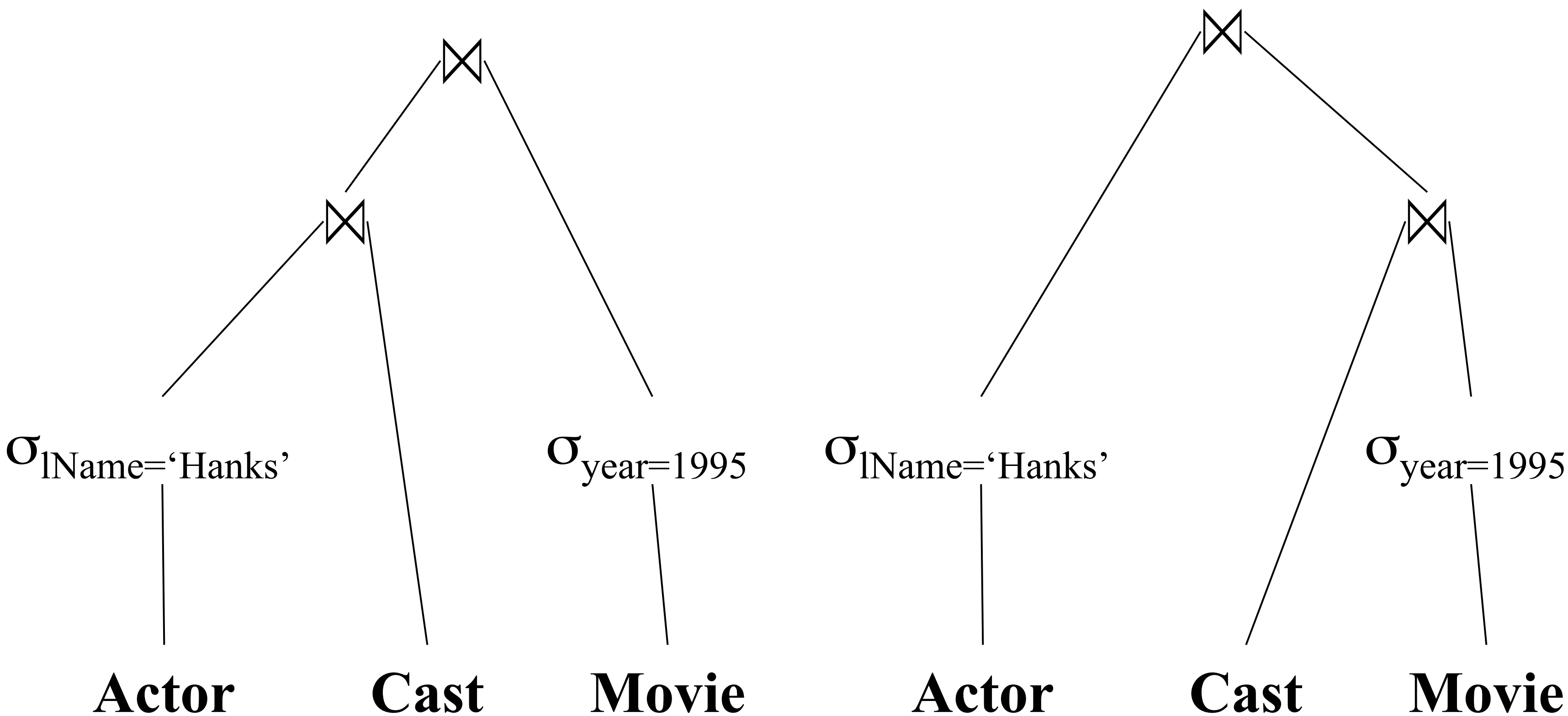
Movie:

id	Name	year
. . .		1995
. . .		

Plan 1: . . . . [in class]

Plan 2: . . . . [in class]

Evaluating Tom Hanks



What Functionality Should a DBMS Support ?

- [in class]

What Functionality Should a DBMS Support ?

1. Data independence
2. Efficient data access
3. Data integrity and security
4. Concurrent access
5. Crash recovery

1. Data Independence

- Separation between:
  - *Physical representation* of the data
  - *Logical view* of the data
- The physical rep may change to improve efficiency (add/drop index, etc)
- Applications not affected: they see only the logical view

Lectures 1,2

2. Efficient Data Access

- Physical data storage: indexes, data clustering
- Query processing: efficient algorithms for accessing/processing the data
- Query optimization: choosing between alternative, equivalent plans

Lectures 6, 7

3. Data Integrity and Security

- Integrity: enforce application constraints during database updates
- Security: access control to the data

Lecture 3

4. Concurrency Control

User 1:

```
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);
```

User 2:

```
X = Read(Account#2);
X.amount = X.amount - 30;
Write(Account#2, X);

Y = Read(Account#3);
Y.amount = Y.amount + 30;
Write(Account#3, Y);
```

What can go wrong ?

Lecture 4

5. Recovery from Crashes

```
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);
```

CRASH !

What can go wrong ?

Lecture 5

Data Management Beyond DBMS

- Other data formats:
  - Semistructured data: XML
  - XPath/XQuery

- Large scale data processing
  - Stream processing
  - Advanced hashing techniques (min-hashes, LSH)
  - Sampling

Lectures 8, 9, 10

(An Example)

Quiz:

- Alice sends Bob in random order all the numbers 1, 2, 3, …, 100000000000000000000
- She does not repeat any number
- But she misses <u>*exactly one number*</u> !
- Help Bob find out which one is missing !

Solved it ? Try this:

- As above, but Alice misses <u>*exactly ten numbers*</u> !

Lectures

1. SQL (today)
2. Database design, Normal Forms
3. Constraints, Views, Security
4. Transactions (recovery)
5. Transactions (concurrency control)
6. Data storage, indexes, physical tuning
7. Query execution and optimization
8. XML/Xpah/Xquery
9. -- 10. Advanced topics

Homeworks

1. SQL [SQL Server] 4/14
2. *Conceptual Design* 4/21
3. SQL App [SQL Server + Postgres] 4/28
4. *Transactions* 5/5
5. Database Tuning [Postgres] 5/19
6. *Optimizations* 5/26
7. XQuery [Galax] 6/2

programming

theory

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 = [login]_P544   Change it !

[See tunneling, MSDNAA]

Then play with IMDB, start working on HW 1

Today: SQL !

- Datatypes in SQL
- Simple Queries in SQL
- Joins
- Subqueries
- Aggregates
- 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

Table name

Attribute names

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

Tuples or rows

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
```

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

"selection" and "projection"

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, 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

Key

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

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 (<u>pname</u>, price, category, manufacturer)

Company (<u>cname</u>, stockPrice, country)

Find all products over $100 manufactured in Japan;
return their names and prices.

Join between Product and Company

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price >= 100
```

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 >= 100
```

PName	Price
SingleTouch	$149.99

In Class

Product (pname, price, category, cname)

Company (cname, stockPrice, country)

Find all Chinese companies that manufacture products in the ‘toy’ category

```
SELECT   cname

FROM

WHERE
```

In Class

Product (<u>pname</u>, price, category, cname)

Company (<u>cname</u>, stockPrice, country)

Find all Chinese companies that manufacture products both in the ‘electronic’ and ‘toy’ categories

```
SELECT   cname

FROM

WHERE
```

In Class

Product (pname, price, category, cname)

Company (cname, stockPrice, country)

Find all Chinese companies that manufacture products both in the ‘electronic’ and ‘toy’ categories

Tuple variables

```
SELECT   z.cname

FROM     Product x, Product y, Company z
WHERE    x.cname=z.cname and y.cname=z.cname and
         x.category='electronic' and y.category='toy'
```

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 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 ?

```
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
```

Returns $R \cap (S \cup T)$
if $S \neq \phi$ and $T \neq \phi$

Subqueries

A subquery (aka *nested* query) may occur in:

1. A SELECT clause
2. A FROM clause
3. A WHERE clause

<u>Rule of thumb</u>: avoid nested queries when possible; sometimes cannot avoid them

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 (later)

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 WHERE 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 <u>some</u> 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

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 <u>some</u> 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

Universal quantifiers

Product (pname, price, company)

Company(cname, city)

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

same as:

Find all cities with companies where all products have price < 100

Universal quantifiers are hard ! ☹

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. Now, 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

Universal quantifiers

Product (pname, price, company)

Company(cname, city)

Find all cities with companies
that make <u>only</u> 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

Rule of Thumb

Non-monotone queries cannot be unnested. In particular, queries with universal 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 <u>some</u> bar that serves <u>some</u> 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 <u>only</u> bars that serves <u>some</u> 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 <u>some</u> bar that serves <u>only</u> 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 <u>only</u> bars that serves <u>only</u> beer they like.

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

Aggregation

```
SELECT  avg(price)
FROM    Product
WHERE   maker='Toyota'
```

```
SELECT  count(*)
FROM    Product
WHERE   year > 1995
```

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
```

Almost the same as Count(*):

- count(category) does not count any category = NULL

We probably want:

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

Grouping and Aggregation

Purchase(product, price, quantity)

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

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

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: grouped attributes and aggregates.

1&2. FROM-WHERE-GROUPBY

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	~~0.5~~	~~50~~
Banana	2	10
Banana	4	10

3. SELECT

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	~~0.5~~	~~50~~
Banana	2	10
Banana	4	10

Product	TotalSales
Bagel	40
Banana	20

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

GROUP BY v.s. Nested Quereis

Purchase(product, price, quantity)

```
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 y.price > 1)
                             AS TotalSales
FROM        Purchase x
WHERE       x.price > 1
```

Why twice ?

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)

```
SELECT      R.A, count(*)
FROM        R
WHERE       R.B < 55
GROUP BY    R.A
```

Always > 0
(Why ?)

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 $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Why ?

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

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

C2 = is any condition on aggregate expressions

General form of Grouping and Aggregation

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

Evaluation steps:

Evaluate FROM-WHERE, apply condition C1

Group by the attributes $a_1,\ldots,a_k$

Apply condition C2 to each group (may have aggregates)

Compute aggregates in S and return the result

Advanced SQLizing

1. INTERSECT and EXCEPT

2. Unnesting Aggregates

3. Finding witnesses

INTERSECT and EXCEPT: not in some DBMS

INTERSECT and EXCEPT:

```
(SELECT R.A, R.B
FROM    R)
   INTERSECT
(SELECT S.A, S.B
FROM    S)
```

=

```
SELECT R.A, R.B
FROM    R
WHERE
    EXISTS(SELECT *
           FROM S
           WHERE R.A=S.A and R.B=S.B)
```

Can unnest.
How ?

```
(SELECT R.A, R.B
FROM    R)
   EXCEPT
(SELECT S.A, S.B
FROM    S)
```

=

```
SELECT R.A, R.B
FROM    R
WHERE
   NOT  EXISTS(SELECT *
               FROM S
               WHERE R.A=S.A and R.B=S.B)
```

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 cname, 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=X.company
                           AND Z.city = X.city)
FROM  Company X
```

```
SELECT X.city, count(*)
FROM Company X, Product Y
WHERE X.cname=Y.company
GROUP BY X.city
```

They are NOT equivalent ! (WHY?)

More Unnesting

Author(login,name)

Wrote(login,url)

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

This is SQL by a novice

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

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(<u>sid</u>, sname)

Product(<u>pid</u>, 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) 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:

```
SELECT Store.sname, x.pname
FROM   Store, Product x
WHERE  Store.sid = x.sid and
       x.price >=
          ALL (SELECT y.price
               FROM Product y
               WHERE Store.sid = y.sid)
```

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*(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

C1 AND C2 = min(C1, C2)

C1 OR C2 = max(C1, C2)

NOT C1 = 1 – C1

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

E.g.
age=20
heigth=NULL
weight=200

Rule in SQL: include only tuples that yield TRUE

Null Values

Unexpected behavior:

```
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(<u>name</u>, 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

Product(<u>name</u>, category)
Purchase(prodName, store)

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

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

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Application

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
```

What's wrong ?

Application

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

Product(name, category)

Purchase(prodName, month, store)

Need to use attribute to get correct zero count

```
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