Introduction to Database Systems CSEP 544

Lecture #1

March 29, 2004

Alon Halevy

1

Staff

- Instructor: Alon Halevy
  - Allen Center, Room 576, alon@cs.washington.edu
  - Office hours: Just before class (or by email)
- TA: Stefan (Stebbi) Sigurdsson
  - stebbi@cs.washington.edu
  - Office hours: TBA

2

Communications

- Web page:

http://www.cs.washington.edu/education/courses/csep544/04sp/

- Mailing list: follow the directions at http://mailman.cs.washington.edu/csenetid/auth/mailman/listinfo/csep544

3

Textbook(s)

Main textbook, available at the bookstore:

- Database Systems: The Complete Book, Hector Garcia-Molina, Jeffrey Ullman, Jennifer Widom

Almost identical, and also available at the bookstore:

- A First Course in Database Systems, Jeff Ullman and Jennifer Widom
- Database Implementation, Hector Garcia-Molina, Jeff Ullman and Jennifer Widom
- Comments on the textbook

4

Other Texts

- Database Management Systems, Ramakrishnan
  - very comprehensive
- Fundamentals of Database Systems, Elmasri, Navathe
  - very widely used
- Foundations of Databases, Abiteboul, Hull, Vianu
  - Mostly theory of databases
- Data on the Web, Abiteboul, Buneman, Suciu
  - XML and other new/advanced stuff

5

Other Required Readings

There will be reading assignments from the Web:

- SQL for Web Nerds, by Philip Greenspun, http://philip.greenspun.com/sql/
- Others, especially for XML

For SQL, a good source of information is the MSDN library (on your Windows machine)

6

Course Structure

- Prerequisites: Data structures course
- Work & Grading:
  - Homework 30% : 3 of them, some light programming.
  - Project: 35% - coming up next.
  - Final: 35% (Discuss date)

7

The Project

- Important component of the course.
- 2 Phases.
- I'll tell you about phase 2 later.
- Phase 1:
  - You build a database application on your own.
  - The domain of the application is inventory of some sort.
  - The application will have a simple web interface.
  - Done by the end of week 4.

8

Today

- Motivation: why do we want databases.
- Overview of database systems
  - Reading assignment from SQL for Web Nerds, by Philip Greenspun, Introduction http://philip.greenspun.com/sql/
- Course Outline.
- Basic elements of SQL

9

What Is a Relational Database Management System ?

Database Management System = DBMS
Relational DBMS = RDBMS

- A program that makes it easy for you to manipulate large amounts of data.
- Frees you from thinking about details. Enables you to focus on your challenges.

10

Where are RDBMS used ?

- Backend for traditional "database" applications
  - Students and courses at a university
  - Bank accounting
  - Airline reservations
  - Movie listings
- Backend for large Websites
- Backend for Web services

11

Example of a Traditional Database Application

Suppose we are building a system
to store the information about:
- students
- courses
- professors
- who takes what, who teaches what

12

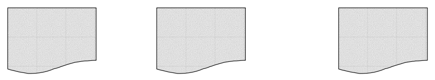
Data Management

- Data management is more than databases.
- Imagine:
  - Complete Traffic Information Availability
  - MyNeededBits Anytime, Anywhere
  - <your favorite visionary application here>
- The techniques we learn are the principles of managing data anywhere.

Can we do it without a DBMS ?

Sure we can! Start by storing the data in files:

students.txt courses.txt professors.txt



Now write C or Java programs to implement specific tasks

Doing it without a DBMS...

- Enroll "Mary Johnson" in "CSE444":

Write a C program to do the following:

```
Read 'students.txt'
Read 'courses.txt'
Find& update the record "Mary Johnson"
Find& update the record "CSE444"
Write "students.txt"
Write "courses.txt"
```

Problems without a DBMS...

- System crashes:

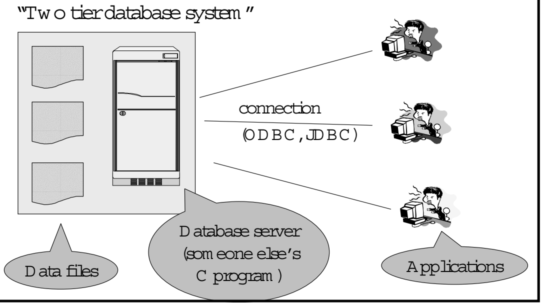
```
Read 'students.txt'
Read 'courses.txt'
Find& update the record "Mary Johnson"
Find& update the record "CSE444"
Write "students.txt"
Write "courses.txt"
```

CRASH !

  - What is the problem ?
- Large data sets (say 50GB)
  - What is the problem ?
- Simultaneous access by many users
  - Need locks: we know them from OS, but now data on disk; and is there any fun to re-implement them ?

Enters a DMBS

"Two tier database system"



Functionality of a DBMS

The programmer sees SQL, which has two components:
- Data Definition Language - DDL
- Data Manipulation Language - DML
  - query language

Behind the scenes the DBMS has:
- Query optimizer
- Query engine
- Storage management
- Transaction Management (concurrency, recovery)

How the Programmer Sees the DBMS

- Start with DDL to create tables:

```
CREATE TABLE Students (
        Name CHAR(30)
        SSN CHAR(9) PRIMARY KEY NOT NULL,
        Category CHAR(20)
) ...
```

- Continue with DML to populate tables:

```
INSERT INTO Students
VALUES('Charles', '123456789', 'undergraduate')
. . . .
```

19

How the Programmer Sees the DBMS

- Tables:

Students:

SSN	Name	Category
123-45-6789	Charles	undergrad
234-56-7890	Dan	grad
	…	…

Takes:

SSN	CID
123-45-6789	CSE444
123-45-6789	CSE444
234-56-7890	CSE142
	…

Courses:

CID	Name	Quarter
CSE444	Databases	fall
CSE541	Operating systems	winter

- Still implemented as files, but behind the scenes can be quite complex

"data independence" = separate logical view from physical implementation

20

Building an Application with a DBMS

- Requirements modeling (conceptual, pictures)
  - Decide what entities should be part of the application and how they should be linked.
- Schema design and implementation
  - Decide on a set of tables, attributes.
  - Define the tables in the database system.
  - Populate database (insert tuples).
- Write application programs using the DBMS
  - way easier now that the data management is taken care of.

21

Transactions

- Enroll "Mary Johnson" in "CSE444":

```
BEGIN TRANSACTION;

INSERT INTO Takes
  SELECT Students.SSN, Courses.CID
  FROM Students, Courses
  WHERE Students.name = 'Mary Johnson' and
        Courses.name = 'CSE444'

-- More updates here....

IF everything-went-OK
  THEN COMMIT;
ELSE ROLLBACK
```

If system crashes, the transaction is still either committed or aborted

22

Transactions

- A transaction = sequence of statements that either all succeed, or all fail
- Transactions have the ACID properties:

A = atomicity

C = consistency

I = independence

D = durability

23

Queries

- Find all courses that "Mary" takes

```
SELECT  C.name
FROM    Students S, Takes T, Courses C
WHERE   S.name="Mary" and
        S.ssn = T.ssn and T.cid = C.cid
```

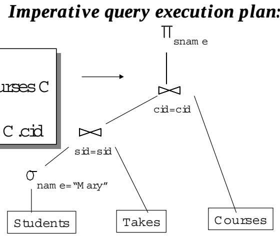
- What happens behind the scene ?
  - Query processor figures out how to answer the query efficiently.

24

Queries, behind the scene

Declarative SQL query → ***Imperative query execution plan:***

```
SELECT C.name
FROM Students S, Takes T, Courses C
WHERE S.name="Mary" and
      S.ssn = T.ssn and T.cid = C.cid
```



The optimizer chooses the best execution plan for a query

Database Systems

- The big commercial database vendors:
  - Oracle
  - IBM (with DB2) bought Informix recently
  - Microsoft (SQL Server)
  - Sybase
- Some free database systems (Unix) :
  - Postgres
  - Mysql
  - Predator
- In CSEP544 we use SQL Server. You may use something else, but you are on your own.

New Trends in Databases

- Object-relational databases
- Main memory database systems
- XML XML XML !
  - Relational databases with XML support
  - Middleware between XML and relational databases
  - Native XML database systems
  - Lots of research here at UW on XML and databases
- Data integration
- Peer to peer, stream data management – still research

The Study of DBMS

- Several aspects:
  - Modeling and design of databases
  - Database programming: querying and update operations
  - Database implementation
- DBMS study cuts across many fields of Computer Science: OS, languages, AI, Logic, multimedia, theory...

Course Outline (may vary slightly)

Part I

- SQL (Chapter 7) and its advanced features.
- Database design (Chapters 2, 3, 7)
- XML, XPath, XQuery
- Data storage, indexes (Chapters 11-13)
- Query execution and optimization (Chapter 15,16)
- Data integration, meta-data management

Table name

Attribute names

The Relational Model (Codd)

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

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

31

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

32

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

33

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

34

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().

35

Tables Explained

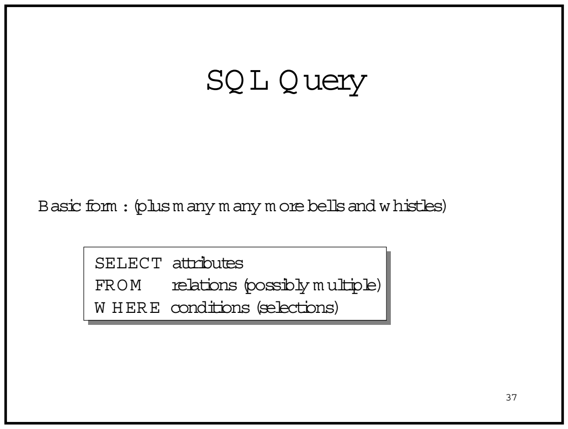
- The schema of a table is the table name and its attributes:

Product(PName, Price, Category, Manfacturer)

- A key is an attribute whose values are unique; we underline a key

Product(<u>PName</u>, Price, Category, Manfacturer)

36

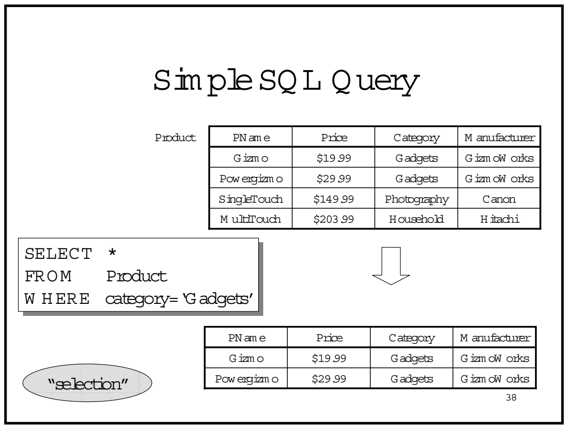


SQL Query

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

```
SELECT  attributes
FROM    relations (possibly multiple)
WHERE   conditions (selections)
```

37



Simple SQL Query

Product

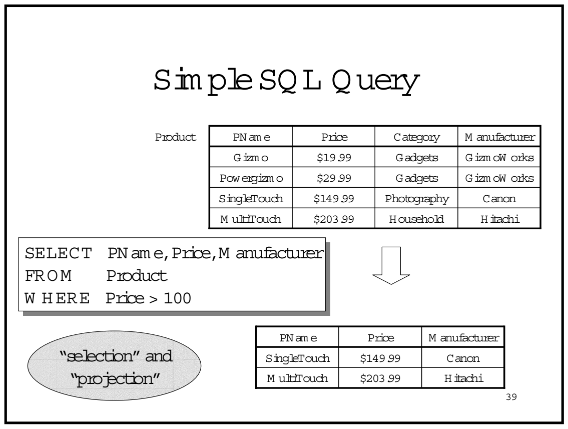
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```

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

"selection"

38



Simple SQL Query

Product

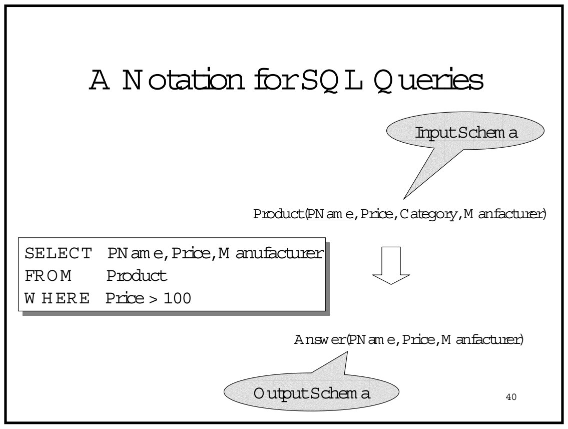
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

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

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

"selection" and "projection"

39



A Notation for SQL Queries

Input Schema

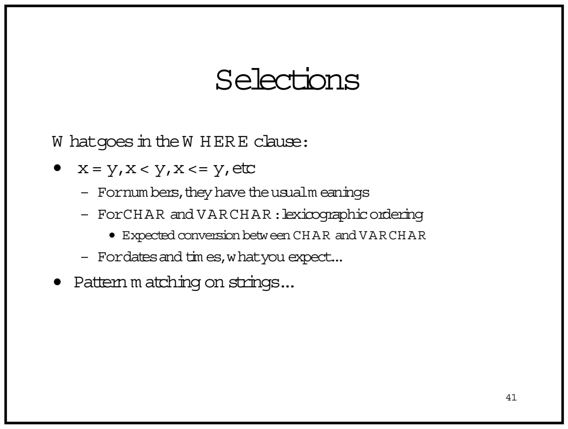
Product(PName, Price, Category, Manfacturer)

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

Answer(PName, Price, Manfacturer)

Output Schema

40

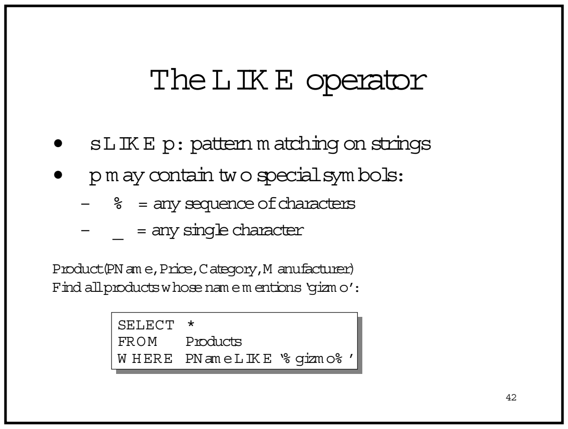


Selections

What goes in the WHERE clause:

- x = y, x < y, x <= y, etc
  - For numbers, 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...

41



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%'
```

42

Eliminating Duplicates

```
SELECT  DISTINCT category
FROM    Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT  category
FROM    Product
```

Category
Gadgets
Gadgets
Photography
Household

43

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.

44

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

?

45

Ordering the Results

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

Category
Gadgets
Household
Photography

Compare to:

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

?

46

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

Cname	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.

Join between Product and Company

```
SELECT  pname, price
FROM    Product, Company
WHERE   manufacturer=cname AND country='Japan'
        AND price <= 200
```

48

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

49

Joins

Product (<u>pname</u>, price, category, manufacturer)
Company (<u>cname</u>, stockPrice, country)

Find all countries that manufacture some product in the 'Gadgets' category.

```
SELECT  country
FROM    Product, Company
WHERE   manufacturer=cname AND category='Gadgets'
```

50

Joins

Product (<u>pname</u>, price, category, manufacturer)
Purchase (buyer, seller, store, product)
Person (<u>persname</u>, 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

52

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
```

53

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)

54

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.

55

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 ∪ {(a1,…,ak)}
return Answer
```

56

First Unintuitive SQLism

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

Looking for R ∩ (S ∪ T)

But what happens if T is empty?

57

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.

58

Union, Intersection, Difference

```
(SELECT name
 FROM   Person
 WHERE  City="Seattle")

  UNION

(SELECT name
 FROM   Person, Purchase
 WHERE  buyer=name AND store="The Bon")
```

Similarly, you can use INTERSECT and EXCEPT.
You must have the same attribute names (otherwise: rename).

59

Conserving Duplicates

```
(SELECT name
 FROM   Person
 WHERE  City="Seattle")

  UNION ALL


(SELECT name
 FROM   Person, Purchase
 WHERE  buyer=name AND store="The Bon")
```

60

Subqueries

A subquery producing a single value:

```
SELECT Purchase.product
FROM   Purchase
WHERE buyer =
        (SELECT name
         FROM   Person
         WHERE  ssn = '123456789');
```

In this case, the subquery returns one value.

If it returns more, it's a run-time error.

61

Can say the same thing without a subquery:

```
SELECT Purchase.product
FROM   Purchase, Person
WHERE buyer = name AND ssn = '123456789'
```

This is equivalent to the previous one when the ssn is a key
and '123456789' exists in the database;
otherwise they are different.

62

Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

```
SELECT Company.name
FROM   Company, Product
WHERE  Company.name=Product.maker
       AND Product.name IN
            (SELECT Purchase.product
             FROM  Purchase
             WHERE Purchase .buyer = 'Joe Blow');
```

Here the subquery returns a set of values: no more
runtime errors.

63

Subqueries Returning Relations

Equivalent to:

```
SELECT Company.name
FROM   Company, Product, Purchase
WHERE  Company.name= Product.maker
       AND Product.name = Purchase.product
       AND Purchase.buyer = 'Joe Blow'
```

Is this query equivalent to the previous one ?

Beware of duplicates !

64

Removing Duplicates

```
SELECT Company.name
FROM   Company, Product, Purchase
WHERE  Company.name= Product.maker
       AND Product.name = Purchase.product
       AND Purchase.buyer = 'Joe Blow'
```

Multiple copies

```
SELECT DISTINCT Company.name
FROM   Company, Product, Purchase
WHERE  Company.name= Product.maker
       AND Product.name = Purchase.product
       AND Purchase.buyer = 'Joe Blow'
```

Single copies

65

Removing Duplicates

```
SELECT DISTINCT Company.name
FROM   Company, Product
WHERE  Company.name= Product.maker
       AND Product.name IN
           (SELECT Purchase.product
            FROM   Purchase
            WHERE Purchase.buyer = 'Joe Blow')
```

```
SELECT DISTINCT Company.name
FROM   Company, Product, Purchase
WHERE  Company.name= Product.maker
       AND Product.name = Purchase.product
       AND Purchase.buyer = 'Joe Blow'
```

Now
they are
equivalent

66

Subqueries Returning Relations

You can also use: s > ALL R
s > ANY R
EXISTS R

Product (pname, price, category, maker)
Find products that are more expensive than all those produced
By "Gizmo-Works"

```
SELECT name
FROM   Product
WHERE price > ALL (SELECT price
                   FROM   Purchase
                   WHERE maker='Gizmo-Works')
```

7

Question for Database Fans and their Friends

- Can we express this query as a single SELECT-FROM-WHERE query, without subqueries ?
- Hint: show that all SFW queries are monotone (figure out what this means). A query with ALL is not monotone

68

Conditions on Tuples

```
SELECT DISTINCT Company.name
FROM    Company, Product
WHERE  Company.name= Product.maker
       AND  (Product.name,price) IN
              (SELECT Purchase.product, Purchase.price)
               FROM   Purchase
               WHERE Purchase.buyer = "Joe Blow");
```

May not work in SQL server...

69

Correlated Queries

Movie (<u>title, year</u>, director, length)
Find movies whose title appears more than once.

correlation

```
SELECT DISTINCT title
FROM   Movie AS x
WHERE  year <> ANY
              (SELECT year
               FROM   Movie
               WHERE title = x.title);
```

Note (1) scope of variables (2) this can still be expressed as single SFW

70

Complex Correlated Query

Product (pname, price, category, maker, year)

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT DISTINCT pname, maker
FROM   Product AS x
WHERE price > ALL (SELECT price
                   FROM   Product AS y
                   WHERE x.maker = y.maker AND y.year < 1972);
```

Powerful, but much harder to optimize !

71

Aggregation

```
SELECT Avg(price)
FROM    Product
WHERE  maker="Toyota"
```

SQL supports several aggregation operations:

SUM, MIN, MAX, AVG, COUNT

72

Aggregation: Count

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

Except COUNT, all aggregations apply to a single attribute

73

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

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

Better:

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

74

Simple Aggregation

Purchase(product, date, price, quantity)

Example 1: find total sales for the entire database

```
SELECT Sum(price * quantity)
FROM   Purchase
```

Example 1': find total sales of bagels

```
SELECT Sum(price * quantity)
FROM   Purchase
WHERE  product = 'bagel'
```

75

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	0.85	15
Banana	10/22	0.52	7
Banana	10/19	0.52	17
Bagel	10/20	0.85	20

76

Grouping and Aggregation

Usually, we want aggregations on certain parts of the relation.

Purchase(product, date, price, quantity)

Example 2: find total sales after 10/1 per product.

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > "10/1"
GROUPBY   product
```

Let's see what this means...

77

Grouping and Aggregation

1. Compute the FROM and WHERE clauses.
2. Group by the attributes in the GROUPBY
3. Select one tuple for every group (and apply aggregation)

SELECT can have (1) grouped attributes or (2) aggregates.

78

First compute the FROM-WHERE clauses (date > “10/1”) then GROUP BY product:

Product	Date	Price	Quantity
Banana	10/19	0.52	17
Banana	10/22	0.52	7
Bagel	10/20	0.85	20
Bagel	10/21	0.85	15

Then, aggregate

Product	TotalSales
Bagel	$29.75
Banana	$12.48

```
SELECT     product, Sum(price*quantity) AS TotalSales
FROM       Purchase
WHERE      date > “10/1”
GROUPBY    product
```

GROUP BY v.s. Nested Queries

```
SELECT     product, Sum(price*quantity) AS TotalSales
FROM       Purchase
WHERE      date > “10/1”
GROUP BY   product
```

```
SELECT DISTINCT x.product, (SELECT Sum(y.price*y.quantity)
                            FROM    Purchase y
                            WHERE x.product = y.product
                                AND y.date > ‘10/1’)
                           AS TotalSales
FROM      Purchase x
WHERE     x.date > “10/1”
```

Another Example

Product	SumSales	MaxQuantity
Banana	$12.48	17
Bagel	$29.75	20

For every product, what is the total sales and max quantity sold?

```
SELECT     product, Sum(price * quantity) AS SumSales
                     Max(quantity) AS MaxQuantity
FROM       Purchase
GROUP BY product
```

HAVING Clause

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

```
SELECT     product, Sum(price * quantity)
FROM       Purchase
WHERE      date > “9/1”
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:

1. Compute the FROM-WHERE part, obtain a table with all attributes in $R_1,\ldots,R_n$
2. Group by the attributes $a_1,\ldots,a_k$
3. Compute the aggregates in C2 and keep only groups satisfying C2
4. Compute aggregates in S and return the result

85

Aggregation

Author(<u>login</u>,name)

Document(<u>url</u>, title)

Wrote(login,url)

Mentions(url,word)

86

- Find all authors who wrote at least 10 documents:
- Attempt 1: with nested queries

This is SQL by a novice

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

87

- 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

No need for DISTINCT: automatically from GROUP BY

88

- Find all authors who have a vocabulary over 10000 words:

```
SELECT    Author.name
FROM      Author, Wrote, Mentions
WHERE     Author.login=Wrote.login AND Wrote.url=Mentions.url
GROUP BY  Author.name
HAVING    count(distinct Mentions.word) > 10000
```

Look carefully at the last two queries: you may be tempted to write them as a nested queries, but in SQL we write them best with GROUP BY

89

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.

90