Lecture 11: SQL

Wednesday, October 18, 2000

Outline

- Simple Queries in SQL (5.1)
- Queries with more than one relation (5.2)
- Subqueries (5.3)
- Duplicates (5.4)

SQL Introduction

Standard language for querying and manipulating data

Structured **Q**uery **L**anguage

Many standards out there: SQL92, SQL2, SQL3, SQL99
Vendors support various subsets of these, but all of what we'll be talking about.

SQL Introduction

Basic form: (many many more bells and whistles in addition)

Select attributes
From relations (possibly multiple, joined)
Where conditions (selections)

Selections

Company(sticker, name, country, stockPrice)

Find all US companies whose stock is > 50:

```
SELECT   *
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Output schema: R(sticker, name, country, stockPrice)

Selections

What you can use in WHERE:

attribute names of the relation(s) used in the FROM.
comparison operators: =, <>, <, >, <=, >=
apply arithmetic operations: stockprice*2
operations on strings (e.g., "||" for concatenation).
Lexicographic order on strings.
Pattern matching: s LIKE p
Special stuff for comparing dates and times.

The **LIKE** operator

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

Company(sticker, name, address, country, stockPrice)
Find all US companies whose address contains "Mountain":

```
SELECT  *
FROM    Company
WHERE   country="USA" AND
        address LIKE "%Mountain%"
```

Projections

Select only a subset of the attributes

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(name, stock price)

Projections

Rename the attributes in the resulting table

```
SELECT   name AS company, stockprice AS price
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(company, price)

Ordering the Results

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
ORDERBY  country, name
```

Ordering is ascending, unless you specify the DESC keyword.

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

Joins

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stockPrice, country)
Person(per-name, phoneNumber, city)

Find names of people living in Seattle that bought gizmo products, and the names of the stores they bought from

```
SELECT   per-name, store
FROM     Person, Purchase
WHERE    per-name=buyer AND city="Seattle"
                        AND product="gizmo"
```

Disambiguating Attributes

Find names of people buying telephony products:

```
SELECT   Person.name
FROM     Person, Purchase, Product
WHERE        Person.name=buyer
         AND  product=Product.name
         AND  Product.category="telephony"
```

Product (name, price, category, maker)
Purchase (buyer, seller, store, product)
Person(name, phoneNumber, city)

Tuple Variables

Find pairs of companies making products in the same category

```
SELECT  product1.maker, product2.maker
FROM    Product AS product1, Product AS product2
WHERE       product1.category=product2.category
        AND  product1.maker <> product2.maker
```

Product (name, price, category, maker)

Tuple Variables

•Tuple variables introduced automatically by the system:

Product (name, price, category, maker)

```
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.

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 U {(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 U {(a1,…,ak)
return Answer
```

Doesn't impose any order !
Like Datalog

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

3. Translation to Datalog: one rule

$Answer(a_1,\ldots,a_k) \leftarrow R_1(x_{11},\ldots,x_{1p}),\ldots,R_n(x_{n1},\ldots,x_{np})$, Conditions

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

4. Translation to Relational algebra:

$\Pi_{a1,\ldots,ak}$ ($\sigma_{\text{Conditions}}$ (R1 x R2 x … x Rn))

Select-From-Where queries are precisely Select-Project-Join

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?

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).

Exercises

Product (pname, price, category, maker)
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 did not buy French products
Ex #4: Find names of people who bought American products and they live in Seattle.
Ex #5: Find people who bought stuff from Joe or bought products from a company whose stock prices is more than $50.

Subqueries

A subquery producing a single tuple:

```
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.

Can say the same thing without a subquery:

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

Is this query equivalent to the previous one ?

Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

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

Here the subquery returns a set of values

Subqueries Returning Relations

Equivalent to:

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

Is this query equivalent to the previous one ?

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")
```

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

Conditions on Tuples

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

Correlated Queries

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

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

correlation

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

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  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 !

Removing Duplicates

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

Conserving Duplicates

The UNION, INTERSECTION and EXCEPT operators operate as sets, not bags.

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

    UNION  ALL


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