Introduction to Database Systems

CSE 444

Lecture #3
Jan 10 2001

Announcements

- **Special Lecture**
  - **At Sieg 134 on January 19th from 330-450PM**
  - **Topic: Building SQL Applications**
  - **Important For**
    - **Programming Assignment**
    - **Course Project**
- **Form Groups for Course Project NOW**
- **Homework Due in a week**
- **Final: Check Schedule**

2

SQL

Reading: Sec 5 (all subsections, except 5.10)

3

Selection and Projection

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

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

4

Removing Duplicates

Product(pid, name, maker, category, price)

```
SELECT   DISTINCT category
FROM     Product
WHERE    price > 100
```

5

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'
```

6

Grouping, Aggregation

Purchase(product, date, price, quantity)

Example 2: **find total sales after 9/1 per product.**

SELECT product, Sum(price*quantity) AS TotalSales
FROM Purchase
WHERE date > "9/1"
GROUPBY product

7

First compute the relation (date > "9/1") 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

8

Then, aggregate

Product	TotalSales
Bagel	$29.75
Banana	$12.48

SELECT product, Sum(price*quantity) AS TotalSales
FROM Purchase
WHERE date > "9/1"
GROUPBY product

9

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

10

Group By and Having

Purchase(product, date, price, quantity)

Example 2: **find total sales after 9/1 per product.**

SELECT product, Sum(price*quantity) AS TotalSales
FROM Purchase
WHERE date > "9/1"
GROUPBY product
HAVING Sum(quantity) > 10

11

Queries With GROUP BY and HAVING

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
GROUP BY *grouping-list*
HAVING *group-qualification*

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
  - The attribute list (i) must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group,* and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Conceptual Evaluation

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary'* fields are deleted, as before.
- The remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups.
- One answer tuple is generated per qualifying group.

Find the age of the youngest sailor with age ≥ 18, for each rating with at least 2 <u>such</u> sailors

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	
7	35.0

Answer relation

- Only S.rating and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `*unnecessary'*.
- 2nd column of result is unnamed. (Use AS to name it.)

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"
```

15

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

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:

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

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

17

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
```

18

Example Instances

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

- We will use these instances of the Sailors and Reserves relations in our examples.

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Example of Conceptual Evaluation

```
SELECT  S.sname
FROM    Sailors S1, Reserves R1
WHERE   S1.sid=R1.sid AND R1.bid=103
```

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

A Note on Range Variables

- Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND bid=103
```

OR

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid
        AND bid=103
```

It is good style, however, to use range variables always!

Find sailors who've reserved at least one boat

```
SELECT  S.sid
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid
```

- Would adding DISTINCT to this query make a difference?

SQL is Tricky!

```
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?

23

Nested Queries

Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN  (SELECT  R.sid
                   FROM  Reserves R
                   WHERE  R.bid=103)
```

- A WHERE clause can itself contain an SQL query!
- To find sailors who've *not* reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Nested Queries with Correlation

Find names of sailors who've reserved boat #103:

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
              FROM Reserves R
              WHERE R.bid=103 AND S.sid=R.sid)
```

- EXISTS is another set comparison operator, like IN.
- If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103. (UNIQUE checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by *R.bid*?)
- Illustrates why, in general, subquery must be re-computed for each Sailors tuple.

More on Set-Comparison Operators

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* SOME, *op* ALL

Example: 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

27

Example: 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 ?

28

Example: 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")
```

29

Example: 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");
```

30

Example: Correlated Queries

Movie (<u>title</u>, <u>year</u>, 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

31

Example: 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);
```

32

Example: 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");
```

33

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).

34

Find sid's of sailors who've reserved a red <u>or</u> a green boat

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
  AND (B.color='red' OR B.color='green')
```

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND
R.bid=B.bid
          AND B.color='red'
UNION
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND
R.bid=B.bid
          AND B.color='green'
```

Union All Etc.

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")
```

36

Defining Views

Views are relations, except that they are not physically stored.

They are used mostly in order to simplify complex queries and to define conceptually different views of the database to different classes of users.

View: purchases of telephony products:

```
CREATE VIEW telephony-purchases AS
  SELECT product, buyer, seller, store
  FROM Purchase, Product
  WHERE Purchase.product = Product.name
        AND Product.category = "telephony"
```

37

A Different View

```
CREATE VIEW Seattle-view AS

      SELECT buyer, seller, product, store
      FROM   Person, Purchase
      WHERE  Person.city = "Seattle"  AND
             Person.name = Purchase.buyer
```

We can later use the views:

```
    SELECT name, store
    FROM   Seattle-view, Product
    WHERE  Seattle-view.product = Product.name  AND
           Product.category = "shoes"
```

What's really happening when we query a view??

38

What Happens When We Query a View ?

```
SELECT  name, Seattle-view.store
FROM    Seattle-view, Product
WHERE   Seattle-view.product = Product.name  AND
          Product.category = "shoes"
                    ↓
SELECT  name, Purchase.store
FROM    Person, Purchase, Product
WHERE   Person.city = "Seattle"   AND
        Person.name = Purchase.buyer   AND
        Purchase.poduct = Product.name  AND
        Product.category = "shoes"
```

39

Null Values and Outerjoins

⌘If x=Null then 4*(3-x)/7 is still NULL

⌘If x=Null then x="Joe" is UNKNOWN

⌘Three boolean values:
- FALSE = 0
- UNKNOWN = 0.5
- TRUE = 1

40

Null Values and Outerjoins

⌘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)
```

Rule in SQL: include only tuples that yield TRUE

41

Null Values and Outerjoins

Unexpected behavior:

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

Some Persons are not included !

42

Null Values and Outerjoins

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

43

Null Values and Outerjoins

Explicit joins in SQL:
Product(name, category)
Purchase(prodName, store)

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

Same as:
SELECT Product.name, Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName

But Products that never sold will be lost !

44

Null Values and Outerjoins

Left outer joins in SQL:
Product(name, category)
Purchase(prodName, store)

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

45

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	-

46