Lecture 12: SQL

Friday, October 20, 2000

Outline

- Duplicates (5.4)
- Aggregation (5.5)
- Database Modifications (5.6)
- Defining a Relation Schema in SQL (5.7)
- View Definitions (5.8)

Removing Duplicates

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

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

Conserving Duplicates

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

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

   UNION  ALL                ←—— keeps duplicates

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

Aggregation

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

SQL supports several aggregation operations:

SUM, MIN, MAX, AVG, COUNT

Aggregation: Count

Except COUNT, all aggregations apply to a single attribute

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

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

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

Better:

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

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'
```

Simple Aggregations

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

Grouping and Aggregation

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

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
```

Grouping and Aggregation

1. Compute the relation (I.e., the FROM and WHERE).
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.

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

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
```

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

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

Modifying the Database

We have 3 kinds of modifications: insertion, deletion, update.

Insertion: general form --

```
INSERT  INTO  R(A1,…., An)  VALUES  (v1,…., vn)
```

•If we don't provide all the attributes of R, they will be filled with NULL.

•We can drop the attribute names if we're providing all of them in order.

Modifying the Database

Insert a new purchase to the database:

```
INSERT   INTO   Purchase(buyer, seller, product, store)
         VALUES  ("Joe", "Fred", "gizmo", "GizmoStore")
```

More Interesting Insertions

```
INSERT   INTO   PRODUCT(name)

    SELECT  DISTINCT  product
    FROM  Purchase
    WHERE   product  NOT IN
            (SELECT  name
             FROM     Product)
```

The query replaces the VALUES keyword.

Note the order of querying and inserting.

Deletions

```
DELETE    FROM    PURCHASE

WHERE    seller = "Joe"   AND
         product = "Brooklyn Bridge"
```

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.

Updates

```
UPDATE   PRODUCT
SET    price = price/2
WHERE  Product.name  IN
           (SELECT product
            FROM    Sales
            WHERE  Date = today);
```

Data Definition in SQL

So far, SQL operations on the data.
Data Manipulation Language (DML)

Data definition: defining the schema.
Data Definition Language (DDL)

- Define data types
- Create/delete/modify tables
- Create/delete indexes

Data Types in SQL

- Character strings (fixed of varying length)
- Bit strings (fixed or varying length)
- Integer (SHORTINT)
- Floating point
- Dates and times

Domains will be used in table declarations.

To reuse domains:

```
CREATE DOMAIN  address  AS  VARCHAR(55)
```

Creating Tables

```
CREATE    TABLE Person(

        name                    VARCHAR(30),
        social-security-number  INTEGER,
        age                     SHORTINT,
        city                    VARCHAR(30),
        gender                  BIT(1),
        Birthdate               DATE

);
```

Deleting or Modifying a Table

Deleting: DROP Person;

Altering:

```
ALTER TABLE   Person
      ADD   phone  CHAR(16);

ALTER  TABLE   Person
      DROP  age;
```

Default Values

The default of defaults: NULL

Specifying default values:

```
CREATE    TABLE Person(
name                    VARCHAR(30),
social-security-number  INTEGER,
age                     SHORTINT  DEFAULT 100,
city                    VARCHAR(30) DEFAULT  "Seattle",
gender                  CHAR(1)   DEFAULT  "?",
birthdate               DATE)
```

Indexes

REALLY important to speed up query processing time.

Suppose we have a relation

Person (name, ssn, age, city)

An index on "ssn" enables us to fetch a tuple for a given ssn *very* efficiently (not have to scan the whole relation).

The problem of deciding which indexes to put on the relations is very hard! (it's called: physical database design).

Creating Indexes

```
CREATE INDEX  ssnIndex ON Person(ssn)
```

Indexes can be created on more than one attribute:

```
CREATE INDEX doubleindex ON
            Person (age, city)
```

Why not create indexes on everything?

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"
```

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??

Updating Views

How can I insert a tuple into a table that doesn't exist?

```
CREATE VIEW  bon-purchase  AS
   SELECT   store, seller, product
   FROM     Purchase
   WHERE    store = "The Bon Marche"
```

If we make the following insertion:

```
INSERT INTO  bon-purchase
      VALUES  ("the Bon Marche", Joe, "Denby Mug")
```

We can simply add a tuple
("the Bon Marche", Joe, NULL, "Denby Mug")
to relation Purchase.

Non-Updatable Views

```
CREATE VIEW  Seattle-view  AS

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

How can we add the following tuple to the view?

(Joe, "Shoe Model 12345", "Nine West")