Lecture 4: SQL

Wednesday, April 10, 2002

Outline

- SQL Queries:
  - Aggregate operators 5.5
  - Null values 5.6
- SQL Updates
- SQL Data Definition Language: 3.1, 3.2
- SQL Views: 3.6
- Reading Assignment:
  - Integrity constraints 3.2, 3.3, 5.11
  - Embedded SQL, ODBC/JDBC: 5.7, 5.10

Aggregation

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

SQL supports several aggregation operations:

SUM, MIN, MAX, AVG, COUNT

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

Except COUNT, all aggregations apply to a single attribute

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

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

same as Count(*)

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 10/1 per product.**

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > "10/1"
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	$R_1,\ldots,R_n$
WHERE	C1
GROUP BY	$a_1,\ldots,a_k$
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

More Aggregation Examples

Author(login,name)

Document(url, title)

Wrote(login,url)

Mentions(url,word)

More Aggregation Examples

- Find all authors who wrote at least 10 documents:

```
Select author.name
From author, wrote
Where author.login=wrote.login
Groupby author.name
Having count(wrote.url) > 10
```

More Aggregation Examples

- Find all authors who have a vocabulary over 10000:

```
Select author.name
From author, wrote, mentions
Where author.login=wrote.login and wrote.url=mentions.url
Groupby author.name
Having count(distinct mentions.word) > 10000
```

Null Values and Outerjoins

- If x=Null then `4*(3-x)/7` is still NULL
- If x=Null then `x="Joe"` is UNKNOWN
- How do we evaluate SQL queries on data with NULLs ?

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

Null Values and Outerjoins

Ternary Logic:

FALSE	= 0
UNKNOWN	= 0.5
TRUE	= 1

C1 AND C2	=	min(C1, C2)
C1 OR C2	=	max(C1, C2)
NOT C1	=	1 – C1

Rule in SQL: include only tuples that yield TRUE

Null Values and Outerjoins

Unexpected behavior:

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

Some Persons are not included !

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

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
```

Products that never sold will be lost !

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
```

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

Outer Joins

- Left outer join:
  - Include the left tuple even if there's no match
- Right outer join:
  - Incluce 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

Updates in SQL

- Insertions
- Deletions
- Updates

They are called “modifications” or “updates”

Insertions

General form:

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

Example: Insert a new purchase to the database:

```
INSERT INTO Purchase(buyer, seller, product, store)
        VALUES ('Joe', 'Fred', 'wakeup-clock-espresso-machine',
                'The Sharper Image')
```

Missing attribute → NULL.
May drop attribute names if give them in order.

Insertions

```
INSERT   INTO   PRODUCT(name)

    SELECT  DISTINCT  Purchase.product
    FROM      Purchase
    WHERE   Purchase.date > “10/26/01”
```

The query replaces the VALUES keyword.
Here we insert *many* tuples into PRODUCT

Insertion: an Example

Product(name, listPrice, category)
Purchase(prodName, buyerName, price)

prodName is foreign key in Product.name

Suppose database got corrupted and we need to fix it:

Product

name	listPrice	category
gizmo	100	gadgets

Purchase

prodName	buyerName	price
camera	John	200
gizmo	Smith	80
camera	Smith	225

Task: insert in Product all prodNames from Purchase

Insertion: an Example

```
INSERT INTO Product(name)

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

name	listPrice	category
gizmo	100	Gadgets
camera	-	-

Insertion: an Example

```
INSERT   INTO   Product(name, listPrice)

 SELECT  DISTINCT  prodName, price
 FROM  Purchase
 WHERE   prodName  NOT IN (SELECT  name FROM  Product)
```

name	listPrice	category
gizmo	100	Gadgets
camera	200	-
camera ??	225 ??	-

← Depends on the implementation

Deletions

Example:

```
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

Example:

```
UPDATE   PRODUCT
SET    price = price/2
WHERE  Product.name  IN
                (SELECT product
                  FROM    Purchase
                  WHERE  Date =‘Oct, 25, 1999’);
```

Data Definition in SQL

So far we have see the ***Data Manipulation Language***, DML
Next: ***Data Definition Language***, DDL

Data types

Data definition

- Create tables
- Delete tables
- Modify table schema

Indexes: to improve peformance

Data Types in SQL

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

Domains (=types) will be used in table declarations.

To reuse domains:
CREATE DOMAIN address AS VARCHAR(55)

Creating Tables

Example:

```
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:

Example:

```
DROP Person;
```

Altering: (adding or removing an attribute).

Example:

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

ALTER  TABLE   Person
        DROP  age;
```

What happens when you make changes to the schema?

Default Values

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
```

The default of defaults: NULL

Indexes

REALLY important to speed up query processing time.

Suppose we have a relation

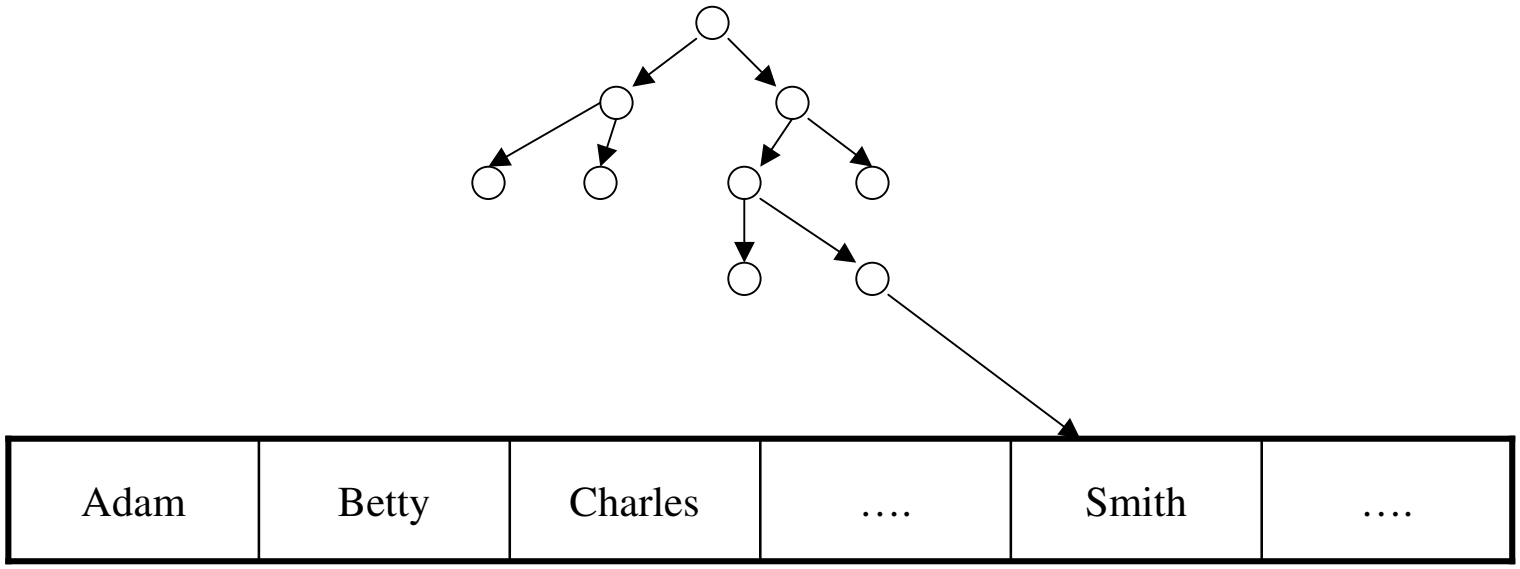
Person (name, age, city)

```
SELECT *
FROM    Person
WHERE   name = “Smith”
```

Sequential scan of the file Person may take long

Indexes

- Create an index on name:



- B+ trees have fan-out of 100s: max 4 levels !

Creating Indexes

Syntax:

```
CREATE INDEX  nameIndex ON Person(name)
```

Creating Indexes

Indexes can be created on more than one attribute:

Example:

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

Helps in:

```
SELECT *
FROM   Person
WHERE age = 55 AND city = “Seattle”
```

But not in:

```
SELECT *
FROM   Person
WHERE city = “Seattle”
```

Creating Indexes

Indexes can be useful in range queries too:

```
CREATE INDEX ageIndex ON  Person (age)
```

B+ trees help in:

```
SELECT *
FROM Person
WHERE age > 25 AND age < 28
```

Why not create indexes on everything?

Views in SQL

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

For presenting different information to different users

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = “Development”
```

Payroll has access to Employee, others only to Developers

A Different View

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

```
CREATE VIEW  Seattle-view  AS

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

We have a new virtual table:
Seattle-view(buyer, seller, product, store)

A Different View

We can later use the view:

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

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”
```

Types of Views

- Virtual views:
  - Used in databases
  - Computed only on-demand – slow at runtime
  - Always up to date
- Materialized views
  - Used in data warehouses
  - Precomputed offline – fast at runtime
  - May have stale data

Updating Views

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

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = "Development"
```

If we make the following insertion:

```
INSERT INTO  Developers
VALUES("Joe", "Optimizer")
```

It becomes:

```
INSERT INTO  Employee
VALUES(NULL, "Joe", NULL, "Optimizer", NULL)
```

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")

We need to add "Joe" to Person first. One copy ? More copies ?