Lecture 04: SQL

Monday, April 2, 2007

Outline

- The Project
- Nulls (6.1.6)
- Outer joins (6.3.8)
- Database Modifications (6.5)

NULLS in SQL

- Whenever we don’t have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if can be null (*nullable* attribute) or not
- How does SQL cope with tables that have NULLs ?

Null Values

- If x= NULL then 4*(3-x)/7 is still NULL
- If x= NULL then x="Joe" is UNKNOWN
- In SQL there are three boolean values:

FALSE	=	0
UNKNOWN	=	0.5
TRUE	=	1

Null Values

- 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)
```

E.g.
age=20
heigth=NULL
weight=200

Rule in SQL: include only tuples that yield TRUE

Null Values

Unexpected behavior:

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

Some Persons are not included !

Null Values

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

Outerjoins

Explicit joins in SQL = “inner joins”:

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 !

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

Application

Compute, for each product, the total number of sales in ‘September’

Product(name, category)

Purchase(prodName, month, store)

```
SELECT Product.name, count(*)
FROM     Product, Purchase
WHERE   Product.name = Purchase.prodName
        and  Purchase.month = ‘September’
GROUP BY Product.name
```

What’s wrong ?

Application

Compute, for each product, the total number of sales in ‘September’

Product(name, category)

Purchase(prodName, month, store)

```
SELECT Product.name, count(*)
FROM     Product LEFT OUTER JOIN Purchase ON
                     Product.name = Purchase.prodName
               and  Purchase.month = ‘September’
GROUP BY Product.name
```

Now we also get the products who sold in 0 quantity

Outer Joins

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

Modifying the Database

Three kinds of modifications

- Insertions
- Deletions
- Updates

Sometimes they are all called “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:

Defines the types.

Data definition: defining the schema.

- Create tables
- Delete tables
- Modify table schema

Indexes: to improve performance

Creating Tables

```
CREATE    TABLE Person(

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

 );
```

Deleting or Modifying a Table

Deleting:

Example:

```
DROP Person;
```

Exercise with care !!

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  INT,
          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

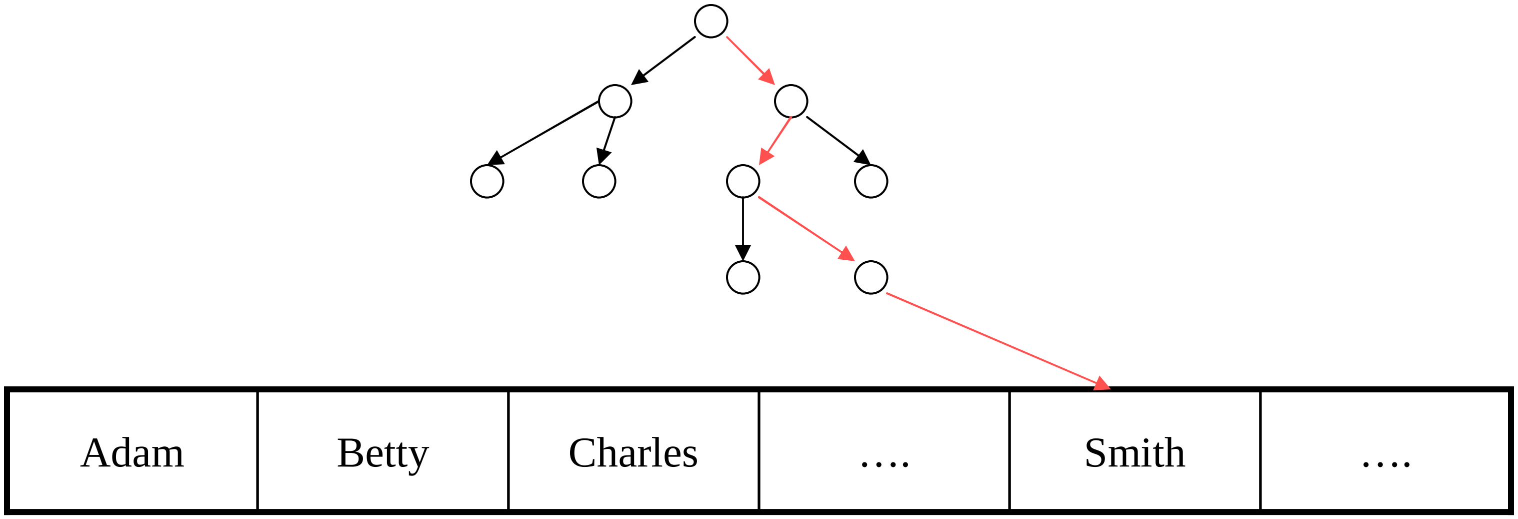
Person (name, age, city)

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

Sequential scan of the file Person may take long

Indexes

- Create an index on name:



Adam	Betty	Charles	….	Smith	….

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

Will discuss in the second half of this course

Creating Indexes

Syntax:

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

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?

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”
```

and even in:

```
SELECT *
FROM    Person
WHERE age = 55
```

But not in:

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

The Index Selection Problem

- Why not build an index on every attribute ? On every pair of attributes ? Etc. ?
- The index selection problem is hard: balance the query cost v.s. the update cost, in a large application workload