Lecture 13: SQL

Monday, October 23, 2000

Outline

- Insertions revisited
- Indexes 5.7.7
- Null Values and Outerjoins 5.9
- Keys and Foreign Keys 6.1, 6.2

Insertions Revisited

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

One expects that prodName is a name occurring in Product

But occasionally the database becomes inconsistent, needs to be fixed:

name	listPrice	category
gizmo	100	gadgets

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

Insertions Revisited

```
INSERT   INTO   Product(name)

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

name	listPrice	category
gizmo	100	Gadgets
camera	-	-

Insertions Revisited

```
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

Indexes

REALLY important to speed up query processing time.

Suppose we have a relation

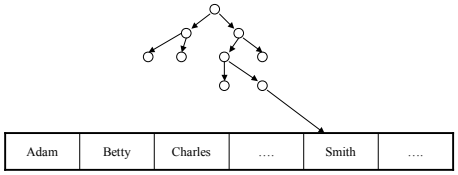
Person (name, age, city)

```
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:

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?

Defining Views

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"
```

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")
```

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 ?

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

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

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
```

But 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	-

Left Outer Joins and XML

```
<Catalog>
    <product> <name> Gizmo </name>
              <store>  Wiz </store>
    </product>
    <product> <name> Camera </name>
              <store>  Ritz </store>
              <store>  Wiz </store>
    </product>
    <product>  <name> OneClick </name>
    </product>
</Catalog>
```

Don't want to lose OneClick just because it never sold !

Outer Joins

- Left outer join:
  - Incluce 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

SQL: Constraints and Triggers

- Chapter 6 Ullman and Widom
- Certain properties we'd like our database to hold
- Modification of the database may break these properties
- Build handlers into the database definition
- Key constraints
- Referential integrity constraints.

Declaring a Primary Keys in SQL

```
CREATE TABLE MovieStar (
   name CHAR(30) PRIMARY KEY,
   address VARCHAR(255),
   gender CHAR(1));
```

OR:

```
CREATE TABLE MovieStar (
   name CHAR(30),
   address VARCHAR(255),
   gender CHAR(1)
   PRIMARY KEY (name));
```

Primary Keys with Multiple Attributes

```
CREATE TABLE MovieStar (
   name CHAR(30),
   address VARCHAR(255),
   gender CHAR(1),
   PRIMARY KEY (name, address));
```

Other Keys

```
CREATE TABLE MovieStar (
   name CHAR(30),
   address VARCHAR(255),
   phone  CHAR(10)  UNIQUE,
   gender CHAR(1),
   petName  CHAR(50),
   PRIMARY KEY (name),
   UNIQUE (gender, petName));
```

Foreign Key Constraints

```
CREATE TABLE ActedIn (
   Name CHAR(30) PRIMARY KEY,
   MovieName CHAR(30)
        REFERENCES Movies(MovieName),
   Year INT);
```

Foreign Key Constraints

- OR

```
CREATE TABLE ActedIn (
   Name CHAR(30) PRIMARY KEY,
   MovieName CHAR(30),
   Year INT,
   FOREIGN KEY MovieName
        REFERENCES Movies(MovieName)
```

- MovieName must be a PRIMARY KEY

How do we Maintain them?

- Given a change to DB, there are several possible violations:
  - Insert new tuple with bogus foreign key value
  - Update a tuple to a bogus foreign key value
  - Delete a tuple in the referenced table with the referenced foreign key value
  - Update a tuple in the referenced table that changes the referenced foreign key value

How to Maintain?

- Recall, ActedIn has FK MovieName...

  Movies(MovieName, year)
  (Fatal Attraction, 1987)

  ActedIn(ActorName, MovieName)
  (Michael Douglas, Fatal Attraction)
  insert: (Rick Moranis, Strange Brew)

How to Maintain?

- Policies for handling the change…
  - Reject the update (default)
  - Cascade (example: cascading deletes)
  - Set NULL
- Can set update and delete actions independently in CREATE TABLE

```
MovieName CHAR(30)
        REFERENCES Movies(MovieName))
                ON DELETE SET NULL
                ON UPDATE CASCADE
```