CSE544
SQL

Monday, April 5, 2004

Announcements

- Homework 1
  - Everybody should have accounts by now
- Project on the Website
  - Please check website, there is work for you !
- Course outline updated on the Website
  - Reading assignments (more to come)
- SQL
  - Lots of materials last lecture and this one !
  - Make sure you understand it

Two Tough Examples

Store(<u>sid</u>, sname)
Product(<u>pid</u>, pname, price, sid)

Find all stores that sell *only* products with price > 100
(Equivalent formulation:
find all stores s.t. all their products have price > 100)


```
SELECT Store.name
FROM   Store, Product
WHERE  Store.sid = Product.sid
GROUP BY  Store.sid, Store.name
HAVING 100 < min(Product.price)
```

Your pick…

```
SELECT Store.name
FROM   Store
WHERE
  100 < ALL (SELECT Product.price
             FROM product
             WHERE Store.sid = Product.sid)
```

```
SELECT Store.name
FROM   Store
WHERE  Store.sid NOT IN
         (SELECT Product.sid
          FROM Product
          WHERE  Product.price <= 100)
```


Two Tough Examples

Store(sid, sname)
Product(pid, pname, price, sid)

For each store, find the Product ID of its most expensive product

Two Tough Examples

This is easy but doesn't do what we want:

```
SELECT Store.name, max(Product.price)
FROM   Store, Product
WHERE  Store.sid = Product.sid
GROUP BY  Store.sid
```

Better:

```
SELECT Store.name, x.pid
FROM   Store, Product x
WHERE  Store.sid = x.sid and
       x.price >=
           ALL (SELECT y.price
                FROM Product y
                WHERE Store.sid = y.sid)
```

But may return multiple pids

Two Tough Examples

Finally, choose some pid arbitrarily, if there are many with highest price:

```
SELECT Store.name, max(x.pid)
FROM   Store, Product x
WHERE  Store.sid = x.sid and
       x.price >=
           ALL (SELECT y.price
                FROM Product y
                WHERE Store.sid = y.sid)
GROUP BY Store.name
```

7

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 ?

8

Nulls

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

Name	Age	Height	Weight
Joe Doe	20	NULL	210
. . .			

9

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

10

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

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

11

Null Values

Unexpected behavior:

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

Some Persons are not included !

12

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

13

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 Produ

14

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
```

15

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

16

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

17

Modifying the Database

Three kinds of modifications

- Insertions
- Deletions
- Updates

Sometimes they are all called "updates"

18

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(<u>name</u>, 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');
```

25

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

26

Data Types in SQL

- Characters:
  - CHAR(20) -- fixed length
  - VARCHAR(40) -- variable length
- Numbers:
  - INT, REAL plus variations
- Times and dates:
  - DATE, DATETIME (SQL Server only)
- To reuse domains:
  CREATE DOMAIN address AS VARCHAR(55)

27

Creating Tables

Example:

```
CREATE     TABLE Person(

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

 );
```

28

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?

29

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

30

Indexes

REALLY important to speed up query processing time.

Suppose we have a relation

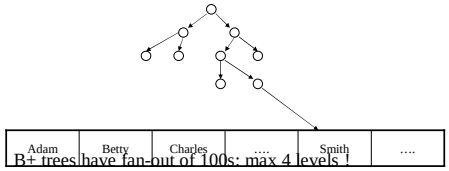
Person (name, age, city)

```
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 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

- We are given a ***workload*** = a set of SQL queries plus how often they run
- What indexes should we build to speed up the workload ?
- FROM/WHERE clauses Ł favor an index
- INSERT/UPDATE clauses Ł discourage an index
- Index selection = normally done by people, recently done automatically (SQL Server)

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

37

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)

38

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"
```

39

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"
```

40

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
  - Pre-computed offline – fast at runtime
  - May have stale data

41

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(ssn, name, department, project, salary)
VALUES(NULL, "Joe", NULL, "Optimizer", NULL)
```

42

Non-Updatable Views

Person(name, city)
Purchase(buyer, seller, product, store)

```
CREATE VIEW  City-Store  AS

     SELECT Person.city, Purchase.store
     FROM     Person, Purchase
     WHERE  Person.name = Purchase.buyer
```

How can we add the following tuple to the view?

("Seattle", "Nine West")

We don't know the name of the person who made the purchase; cannot set to NULL (why ?)

Constraints in SQL

- A constraint = a property that we'd like our database to hold
- The system will enforce the constraint by taking some actions:
  - forbid an update
  - or perform compensating updates

Constraints in SQL

Constraints in SQL:

- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

simplest

Most complex

The more complex the constraint, the harder it is to check and to enforce

Keys

```
CREATE TABLE Product (
        name CHAR(30) PRIMARY KEY,
        category VARCHAR(20))
```

OR:

```
CREATE TABLE Product (
        name CHAR(30),
        category VARCHAR(20)
PRIMARY KEY (name))
```


Keys with Multiple Attributes

```
CREATE TABLE Product (
        name CHAR(30),
        category VARCHAR(20),
        price INT,
    PRIMARY KEY (name, category))
```

Name	Category	Price
Gizmo	Gadget	10
Camera	Photo	20
Gizmo	Photo	30
~~Gizmo~~	~~Gadget~~	~~40~~

Other Keys

```
CREATE TABLE Product (
        productID  CHAR(10),
        name CHAR(30),
        category VARCHAR(20),
        price INT,
        PRIMARY KEY (productID),
        UNIQUE (name, category))
```

There is at most one PRIMARY KEY; there can be many UNIQUE

Foreign Key Constraints

Referential integrity constraints

```
CREATE TABLE Purchase (
      prodName CHAR(30)
            REFERENCES Product(name),
      date DATETIME)
```

prodName is a **foreign key** to Product(name)
name must be a **key** in Product

49

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

50

Foreign Key Constraints

- OR

```
CREATE TABLE Purchase (
      prodName CHAR(30),
      category VARCHAR(20),
      date DATETIME,
      FOREIGN KEY (prodName, category)
        REFERENCES  Product(name, category)
```

- (name, category) must be a PRIMARY KEY

51

What happens during updates ?

Types of updates:

- In Purchase: insert/update
- In Product: delete/update

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

52

What happens during updates ?

- SQL has three policies for maintaining referential integrity:
- Reject violating modifications (default)
- Cascade: after a delete/update do a delete/update
- Set-null set foreign-key field to NULL

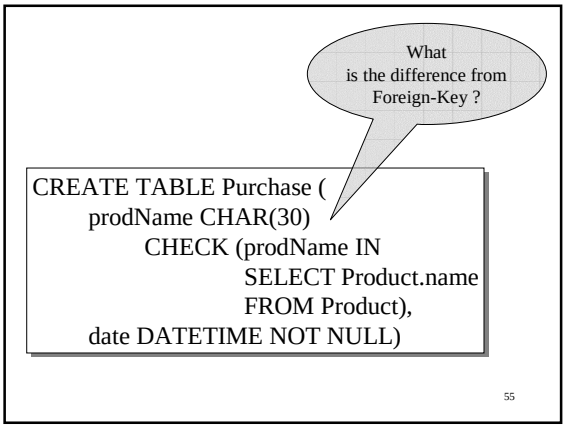
READING ASSIGNEMNT: 7.1.5, 7.1.6

53

Constraints on Attributes and Tuples

- Constraints on attributes:
  NOT NULL -- obvious meaning...
  CHECK condition -- any condition !
- Constraints on tuples
  CHECK condition

54



55

General Assertions

```
CREATE ASSERTION myAssert CHECK
 NOT EXISTS(
      SELECT Product.name
      FROM Product, Purchase
      WHERE Product.name = Purchase.prodName
      GROUP BY Product.name
      HAVING count(*) > 200)
```

56

Final Comments on Constraints

- Can give them names, and alter later
  - Read in the book !!!
- We need to understand exactly *when* they are checked
- We need to understand exactly *what* actions are taken if they fail

57

Embedded SQL

- direct SQL (= ad-hoc SQL) is rarely used
- in practice: SQL is embedded in some application code
- SQL code is identified by special syntax

58

Impedance Mismatch

- Example: SQL in C:
  - C uses int, char[..], pointers, etc
  - SQL uses tables
- Impedance mismatch = incompatible types

59

The Impedance Mismatch Problem

Why not use only one language?

- Forgetting SQL: "we can quickly dispense with this idea" [textbook, pg. 351].
- SQL cannot do everything that the host language can do.

Solution: use cursors

60

Transactions

Address two issues:

- Access by multiple users
  - Remember the "client-server" architecture: one server with many clients
- Protection against crashes

61

Multiple users: single statements

```
Client 1:
        UPDATE Product
        SET Price = Price – 1.99
        WHERE pname = 'Gizmo'

Client 2:
        UPDATE Product
        SET Price = Price*0.5
        WHERE pname='Gizmo'
```

Two managers attempt to do a discount.
Will it work ?

62

Multiple users: multiple statements

```
Client 1:  INSERT INTO SmallProduct(name, price)
                SELECT pname, price
                FROM Product
                WHERE price <= 0.99

           DELETE Product
                WHERE price <=0.99

Client 2:  SELECT count(*)
           FROM Product

           SELECT count(*)
           FROM SmallProduct
```

What's wrong ?

63

Protection against crashes

```
Client 1:
        INSERT INTO SmallProduct(name, price)
                SELECT pname, price
                FROM Product
                WHERE price <= 0.99

        DELETE Product
                WHERE price <=0.99
```

Crash !

What's wrong ?

64

Transactions

- Transaction = group of statements that must be executed atomically
- Transaction properties: ACID
  - ATOMICITY = all or nothing
  - CONSISTENCY = leave database in consistent state
  - ISOLATION = as if it were the only transaction in the system
  - DURABILITY = store on disk !

65

Transactions in SQL

- In "ad-hoc" SQL:
  - Default: each statement = one transaction
- In "embedded" SQL:

  BEGIN TRANSACTION
  [SQL statements]
  COMMIT or ROLLBACK (=ABORT)

66

Transactions: Serializability

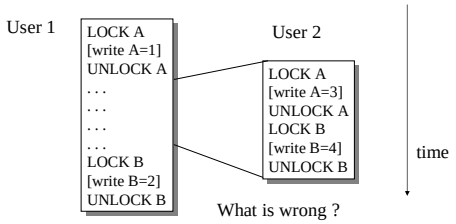
Serializability = the technical term for isolation

- An execution is ***serial*** if it is completely before or completely after any other function's execution
- An execution is ***serializable*** if it equivalent to one that is serial
- DBMS can offer serializability guarantees

67

Serializability

- Enforced with locks, like in Operating Systems !
- But this is not enough:



What is wrong ?

68

Serializability

- Solution: two-phase locking
  - Lock everything at the beginning
  - Unlock everything at the end
- Read locks: many simultaneous read locks allowed
- Write locks: only one write lock allowed
- Insert locks: one per table

69

Isolation Levels in SQL

1. "Dirty reads"
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
2. "Committed reads"
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED
3. "Repeatable reads"
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
4. Serializable transactions (default):
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

Reading assignment: chapter 8.6

70