CIS 560 – Database System Concepts

Lecture 5

SQL

September 6, 2013

Credits for slides: Suciu, Chang, Ullman.

Announcements

- SQL1 assignment due by midnight
  - phpMyAdmin known to hang when executing too many poorly formed queries, e.g.

```
SELECT movie_name, COUNT(aid.actor_id), aid.gender
FROM movie_info, actor_ids AS aid, actor_movies AS amov
WHERE aid.gender = 'female'LIMIT 0, 30;
```

  - Connecting from Linux might work better
  - http://support.cis.ksu.edu/FrequentlyAskedQuestions#FrequentlyAskedQuestions.2BAC8-Misc.ConnectingfromLinux-1
- GTA office hours on Fridays moved to 12:30pm (he also holds office hours on Mondays at 9:30am)
- SQL2 assignment will be posted tonight

Outline

Last time:

- Unnesting aggregates and finding witnesses
- Nulls (Sections 6.1.6 - 6.1.7)

Today:

- Outer joins (Section 6.3.8)
- Views (Sections 8.1, 8.2, 8.3)

Next:

- Constraints (Sections 2.3, 7.1, 7.2)
- E/R Diagrams (Sections 4.1-4.5)

3

Review

- Example of a witness query
- What does a NULL value mean?
- How many Boolean values in SQL?
- How do we evaluate expressions containing NULL?
- Set operations?

4

Outerjoins

Product(name, category)
Purchase(prodName, store)

An "inner join":

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

Same as:

```
SELECT Product.name, Purchase.store
FROM   Product JOIN Purchase ON
                  Product.name = Purchase.prodName
```

But Products that never sold will be lost !

5

Outerjoins

Product(name, category)
Purchase(prodName, store)

If we want the never-sold products, need an "outerjoin":

```
SELECT Product.name, Purchase.store
FROM   Product LEFT OUTER JOIN Purchase ON
                  Product.name = Purchase.prodName
```

6

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

7

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 ?

8

Application

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

Product(name, category)
Purchase(prodName, month, store)

```
SELECT Product.name, count(store)
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 that sold in 0 quantity

9

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 both the left and right tuples even if there's no match

10

Views

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

Useful 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

11

Example

Purchase(customer, product, store)
Product(pname, price)

```
CREATE VIEW  CustomerPrice  AS
      SELECT  x.customer, y.price
      FROM    Purchase x, Product y
      WHERE   x.product = y.pname
```

CustomerPrice(customer, price) "virtual table"

12

Purchase(customer, product, store)
Product(pname, price)
CustomerPrice(customer, price)

We can later use the view:

```
SELECT  u.customer, v.store
FROM    CustomerPrice u, Purchase v
WHERE   u.customer = v.customer  AND
        u.price > 100
```

13

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
  - Indexes *are* materialized views (Section 8.3)

We discuss only virtual views in class

14

Applications of Virtual Views

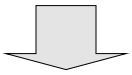
- Physical data independence, e.g.
  - Vertical data partitioning
  - Horizontal data partitioning
- Security
  - The view reveals only what the users are allowed to know

15

Vertical Partitioning

Resumes

SSN	Name	Address	Resume	Picture
234234	Mary	Huston	Clob1…	Blob1…
345345	Sue	Seattle	Clob2…	Blob2…
345343	Joan	Seattle	Clob3…	Blob3…
234234	Ann	Portland	Clob4…	Blob4…



T1

SSN	Name	Address
234234	Mary	Huston
345345	Sue	Seattle
. . .		

T2

SSN	Resume
234234	Clob1…
345345	Clob2…

T3

SSN	Picture
234234	Blob1…
345345	Blob2…

16

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
      SELECT  T1.ssn, T1.name, T1.address,
                T2.resume, T3.picture
      FROM    T1,T2,T3
      WHERE   T1.ssn=T2.ssn and T2.ssn=T3.ssn
```

17

Vertical Partitioning

```
SELECT address
FROM   Resumes
WHERE  name = 'Sue'
```

Which of the tables T1, T2, T3 will be queried by the system ?

18

Vertical Partitioning

When to do this:

- When some fields are large, and rarely accessed
  - E.g. Picture
- In distributed databases
  - Customer personal info at one site, customer profile at another
- In data integration
  - T1 comes from one source
  - T2 comes from a different source

19

Horizontal Partitioning

Customers

SSN	Name	City	Country
234234	Mary	Huston	USA
345345	Sue	Seattle	USA
345343	Joan	Seattle	USA
234234	Ann	Portland	USA
--	Frank	Calgary	Canada
--	Jean	Montreal	Canada

CustomersInHuston

SSN	Name	City	Country
234234	Mary	Huston	USA

CustomersInSeattle

SSN	Name	City	Country
345345	Sue	Seattle	USA
345343	Joan	Seattle	USA

CustomersInCanada

SSN	Name	City	Country
--	Frank	Calgary	Canada
--	Jean	Montreal	Canada

Horizontal Partitioning

```
CREATE VIEW  Customers  AS
    CustomersInHuston
       UNION ALL
    CustomersInSeattle
       UNION ALL
     . . .
```

21

Horizontal Partitioning

```
SELECT name
FROM    Customers
WHERE   city = 'Seattle'
```

Which tables are inspected by the system ?

22

Horizontal Partitioning

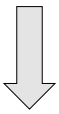
Better:

```
CREATE VIEW  Customers  AS
    (SELECT * FROM CustomersInHuston
     WHERE city = 'Huston')
       UNION ALL
    (SELECT * FROM CustomersInSeattle
     WHERE city = 'Seattle')
       UNION ALL
     . . .
```

23

Horizontal Partitioning

```
SELECT name
FROM   Customers
WHERE  city = 'Seattle'
```



```
SELECT name
FROM   CustomersInSeattle
```

24

Horizontal Partitioning

Applications:

- Optimizations:
  - E.g. archived applications and active applications
- Distributed databases
- Data integration

25

Views and Security

Customers:

Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

Fred is not allowed to see this

Fred is allowed to see this

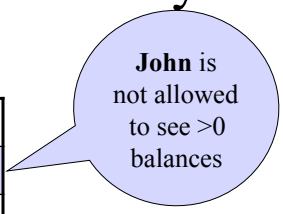
```
CREATE VIEW PublicCustomers
       SELECT Name, Address
       FROM Customers
```

26

Views and Security

Customers:

Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520




```
CREATE VIEW BadCreditCustomers
    SELECT *
    FROM Customers
    WHERE Balance < 0
```

27

Constraints and Triggers

- A *constraint* is a relationship among data elements that the DBMS is required to enforce.
  - Example: key constraints.
- *Triggers* are only executed when a specified condition occurs, e.g., insertion of a tuple.
  - Easier to implement than many constraints.

28

Constraints in SQL

- Keys
- Foreign keys (referential integrity)
- Attribute-level constraints
  - Constrain values of a particular attribute
- Tuple-level constraints
  - Relationships among components
- Global constraints: assertions

simplest

Most complex

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

29

Single Attribute Keys

Product(<u>name</u>, category, price)

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

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

30

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~~

Product(<u>name, category</u>, price)

31

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

32

Foreign Key Constraints

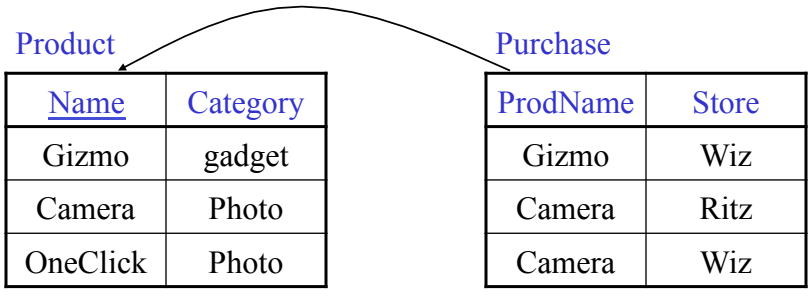
Referential integrity constraints

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

May write just Product (why ?)

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

33



Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

34

Foreign Key Constraints

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

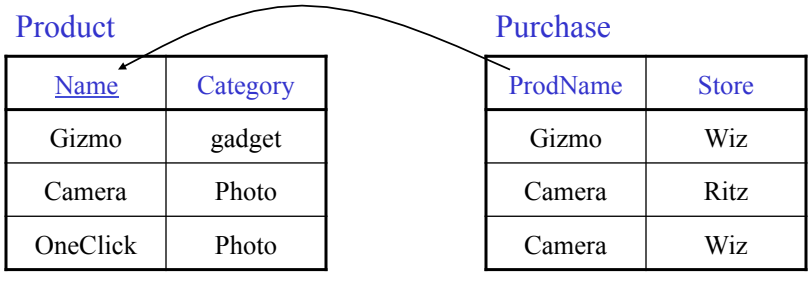
- (name, category) must be a key in Product

35

What happens during updates ?

Types of "problematic" updates:

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



36