CS 480: Database Systems

Lecture 13

February 13,2013

Aggregation with GROUP BY

- INSTRUCTOR(name,dept,salary)
- Retrieve the faculty budget for each department (what they pay in total to their professors).

```
SELECT    dept,SUM(salary) AS budget
FROM      instructor
GROUP BY  dept
```

The attributes in the GROUP BY clause have to also appear in the SELECT clause to appear in the result.

Database Modification Operations

- So far all we've done is: accessing the data (querying).
- Now we can also manipulate the data:
  - Inserting new tuples
  - Deleting tuples
  - Updating tuples

Deletion

- Format:

```
DELETE FROM r
WHERE       P
```

- *r* is the relation from which to choose the tuples to delete.
- *P* is a predicate that selects the tuples to remove.
- If the WHERE clause is omitted, all tuples in *r* are deleted.
- Operates on only one relation.

Deletion

- INSTRUCTOR(name,dept,salary)

Deletion

- INSTRUCTOR(name,dept,salary)
- Example #1: Delete all instructors that are part of the CS department.

Deletion

- INSTRUCTOR(name,dept,salary)
- Example #1: Delete all instructors that are part of the CS department.

```
DELETE FROM instructor
WHERE       dept='CS'
```

Deletion

- INSTRUCTOR(name,dept,salary)
- Example #1: Delete all instructors that are part of the CS department.

```
DELETE FROM instructor
WHERE       dept='CS'
```

- Example #2: Delete all instructors that have a salary between $13,000 and $15000.

Deletion

- INSTRUCTOR(name,dept,salary)
- Example #1: Delete all instructors that are part of the CS department.

```
DELETE FROM instructor
WHERE       dept='CS'
```

- Example #2: Delete all instructors that have a salary between $13,000 and $15000.

```
DELETE FROM instructor
WHERE       salary >= 13000 AND
            salary <= 15000
```

Deletion

- Deletion is from only one relation.
- However, we can reference other relations in SELECT-FROM-WHERE format in nested subqueries inside in the WHERE clause.

Deletion

- STUDENT(id,name,address,GPA)
  ENROL(id,course,grade)

Deletion

- STUDENT(id,name,address,GPA)
  ENROL(id,course,grade)
- Example #3: Delete the records of every student that got an F in CS480.

Deletion

- STUDENT(id,name,address,GPA)
  ENROL(id,course,grade)
- Example #3: Delete the records of every student that got an F in CS480.

```
DELETE FROM student
WHERE       id IN (SELECT id
                   FROM enrol
                   WHERE grade='F' AND
                         course='CS480')
```

Deletion

- INSTRUCTOR(name,dept,salary)
- Consider the following query:

```
DELETE FROM instructor
WHERE       salary < (SELECT AVG(salary)
                      FROM   instructor)
```

Deletion

- INSTRUCTOR(name,dept,salary)
- Consider the following query:

```
DELETE FROM instructor
WHERE       salary < (SELECT AVG(salary)
                      FROM   instructor)
```

Deletion processing note:
Selection of all tuples to be deleted should be done prior to the actual deletion of the tuples. Why?

Insertion

- Two possible ways:
  1. Specify a tuple to be inserted
  2. Write a query whose result is a set of tuples to be inserted.
- Attribute values for inserted tuples must be members of the corresponding attribute's domain.

Insertion

- Insertion by specifying a tuple to be inserted
- COURSE(name,title,dept,credits)

Insertion

- Insertion by specifying a tuple to be inserted
- COURSE(name,title,dept,credits)

```
INSERT INTO course
   VALUES('CS480','Database Systems','CS',4)
```

Insertion

- Insertion by specifying a tuple to be inserted
- COURSE(name,title,dept,credits)

```
INSERT INTO course
   VALUES('CS480','Database Systems','CS',4)
```

Order of values should be based in the order that the schema was defined.
If order is unknown then one can specify the order explicitly:

Insertion

- Insertion by specifying a tuple to be inserted
- COURSE(name,title,dept,credits)

```
INSERT INTO course
   VALUES('CS480','Database Systems','CS',4)
```

Order of values should be based in the order that the schema was defined.
If order is unknown then one can specify the order explicitly:

```
INSERT INTO course(name,title,dept,credits)
   VALUES('CS480','Database Systems','CS',4)
```

Insertion

- Insertion by specifying a tuple to be inserted
- COURSE(name,title,dept,credits)

```
INSERT INTO course
   VALUES('CS480','Database Systems','CS',4)
```

Order of values should be based in the order that the schema was defined.
If order is unknown then one can specify the order explicitly:

```
INSERT INTO course(name,title,dept,credits)
   VALUES('CS480','Database Systems','CS',4)

 INSERT INTO course(name,dept,credits,title)
    VALUES('CS401','CS',4,'Algorithms')
```

Insertion

- Insertion by writing a query whose result is a set of tuples to be inserted.

Insertion

- Insertion by writing a query whose result is a set of tuples to be inserted.
- INSTRUCTOR(name,dept,salary)

  STUDENT(id,name,address,dept,tot_credit)

Insertion

- Insertion by writing a query whose result is a set of tuples to be inserted.
- INSTRUCTOR(name,dept,salary)

  STUDENT(id,name,address,dept,tot_credit)
- Suppose we want to make every CS student an instructor with a salary of $18,000 if they have earned more than 150 credits.

Insertion

- Insertion by writing a query whose result is a set of tuples to be inserted.
- INSTRUCTOR(name,dept,salary)

  STUDENT(id,name,address,dept,tot_credit)
- Suppose we want to make every CS student an instructor with a salary of $18,000 if they have earned more than 150 credits.

```
INSERT INTO instructor
    SELECT name,dept,18000
    FROM   student
    WHERE  dept='CS' AND tot_credit > 150
```

Updates

- Used for changing a value in a tuple without changing all values in the tuple.
- We can choose which tuples to update.
- Format:

```
UPDATE r
SET    A1=a1,A2=a2,…,An=an
WHERE  P
```

- A_i's are attributes, a_i's are arithmetic expressions, P is a predicate.

Updates

- INSTRUCTOR(name,dept,salary)
- Annual salary increases need to be made. All instructor salaries will be increased by 5%.

Updates

- INSTRUCTOR(name,dept,salary)
- Annual salary increases need to be made. All instructor salaries will be increased by 5%.

```
UPDATE instructor
SET    salary = salary*1.05
```

Updates

- INSTRUCTOR(name,dept,salary)
- Annual salary increases need to be made. All instructors with salary less than $70,000 will have their salaries increased by 5%.

Updates

- INSTRUCTOR(name,dept,salary)
- Annual salary increases need to be made. All instructors with salary less than $70,000 will have their salaries increased by 5%.

```
UPDATE instructor
SET    salary = salary*1.05
WHERE  salary < 70000
```

Updates

- INSTRUCTOR(name,dept,salary)
- All instructors that earn over $100,000 will get a 3% raise. All other instructors will get a 5% raise.

Updates

- INSTRUCTOR(name,dept,salary)
- All instructors that earn over $100,000 will get a 3% raise. All other instructors will get a 5% raise.

```
UPDATE instructor
SET    salary = salary*1.03
WHERE  salary > 100000

UPDATE instructor
SET    salary = salary*1.05
WHERE  salary <= 100000
```

Updates

- INSTRUCTOR(name,dept,salary)
- All instructors that earn over $100,000 will get a 3% raise. All other instructors will get a 5% raise.

```
UPDATE instructor
SET    salary = CASE
                 WHEN salary <= 100000 then salary*1.05
                 ELSE salary*1.03
                END
```

Updates with CASE construct

- In general:

```
UPDATE r
SET    A = CASE
             WHEN P1 then a1
             WHEN P2 then a2
             …
             WHEN Pn then an
             ELSE a0
           END
```

Updates

- One could also update a value with the result of a scalar query (one that returns just one value).
- STUDENT(id,name,address,tot_credit)

  COURSE(name,title,credits)

  ENROL(id,course_name,grade)

```
UPDATE student AS s
SET    tot_credit = (
           SELECT SUM(credits)
           FROM   enrol NATURAL JOIN course
           WHERE  s.id=enrol.id AND
                  enrol.grade <> 'F')
```

Data Definition Language

- SQL DDL provides commands for:
  - Defining relation schemas
  - Deleting relations
  - Modifying relation schemas
- When defining a relation in SQL we can specify:
  - The schema for each relation
  - Types of values associated with each attribute
  - Other constraints (integrity, access, storage)

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with *p* digits and with *d* of the digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with *p* digits and with *d* of the digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- Many others not covered here

CREATE TABLE construct

- SQL relations are defined using the CREATE TABLE command:

```
CREATE TABLE r
        (A1 D1,
         A2 D2,
         …
         An Dn,
         <integrity_constraint1>,
         …
         <integrity_constraintk>)
```

Some Integrity Constraints

- `PRIMARY KEY(`B_1`,`B_2`,…,`B_m`)`
  - B's must be a subset of the already specified attributes (A's).
  - Requires the key attributes to be unique.
  - Optional, but it's generally good practice to specify one.
- `FOREIGN KEY(`C_1`,`C_2`,…,`C_d`) REFERENCES s`
  - The values of these C's should correspond to values of these attributes of some tuple in relation s.
- SQL will prevent any update to the database that violates an integrity constraint.

SQL Beer-drinker Database

Relational Schema:
DRINKER(d-name,d-city), BAR(ba-name,ba-city), BEER(be-name,type),
FREQUENTS(d-name,ba-name), SERVES(ba-name,be-name),
LIKES(d-name,be-name)

SQL schema definition

```
CREATE TABLE drinker
      (d_name VARCHAR(30),
       d_city VARCHAR(20),
       PRIMARY KEY (d_name));
CREATE TABLE bar
      (ba_name VARCHAR(30),
       ba_city VARCHAR(20),
       PRIMARY KEY (ba_name));
CREATE TABLE beer
      (be_name VARCHAR(30),
       type    VARCHAR(20),
       PRIMARY KEY (be_name));
```

SQL Beer-drinker Database

Relational Schema:
DRINKER(d-name,d-city), BAR(ba-name,ba-city), BEER(be-name,type),
FREQUENTS(d-name,ba-name), SERVES(ba-name,be-name),
LIKES(d-name,be-name)

SQL schema definition

```
CREATE TABLE frequents
      (d_name  VARCHAR(30),
       ba_name VARCHAR(30),
       FOREIGN KEY(d_name) REFERENCES drinker,
       FOREIGN KEY(ba_name) REFERENCES bar);
CREATE TABLE serves
      (ba_name VARCHAR(30),
       be_name VARCHAR(30),
       FOREIGN KEY(be_name) REFERENCES beer,
       FOREIGN KEY(ba_name) REFERENCES bar);
CREATE TABLE likes
      (d_name  VARCHAR(30),
       be_name VARCHAR(30),
       FOREIGN KEY(be_name) REFERENCES beer,
       FOREIGN KEY(d_name) REFERENCES drinker);
```

CHECK clause

- An additional integrity constraint.
- When applied to an attribute declaration it specifies a predicate that must be satisfied by every tuple in the relation.
- Common use is to ensure that attribute values satisfy specified conditions (creates a powerful type system)
- Syntax: `CHECK(P)`

CHECK clause

```
CREATE TABLE beer
      (be_name VARCHAR(30),
       type    VARCHAR(20),
       PRIMARY KEY (be_name),
       CHECK (type IN ('ale','lager',
                       'stout','pilsner',
                       'porter','bitter',
                       'other')));
```

CHECK clause

```
CREATE TABLE employee
      (id      CHAR(10)
       name    VARCHAR(30),
       salary NUMERIC(10,2),
       PRIMARY KEY (id),
       CHECK (salary > 0));
```

Deleting Tables

- To remove a relation from an SQL database we use the DROP TABLE command.
- Syntax: `DROP TABLE r`

Altering Tables

- We can use the ALTER TABLE command to change the scheme of a relation by:
  - Adding attributes to the relation
    - Syntax: `ALTER TABLE r ADD A D;`
    - Where r is an existing relation, A is the name of the new attribute and D is the domain of A.
    - All values for the new attribute are initially assigned the special value `NULL`.
  - Removing attributes from the relation
    - Syntax: `ALTER TABLE r DROP A;`
    - Where r is an existing relation and A is an attribute of r.

Relational Calculus

Tuple Relational Calculus

- Relational algebra expressions provide a sequence of procedure that generates the answer to our query.
- Tuple Relational Calculus in contrast is a formal query language that is **nonprocedural**.
- It describes the desired information without giving a specific procedure for obtaining the information.

Tuple Relational Calculus

- General format of a relational calculus expression:

{ *x* | <condition>}

Example #1

- student1(STUDENT)
- student2(STUDENT)
- STUDENT = {id,name,major,GPA}
- RA Query: student1 $\cup$ student2
- Calculus:

$$\{ y \mid y \in \text{student1} \vee y \in \text{student2} \}$$

Example #2

- student(STUDENT)
- STUDENT = {id,name,major,GPA}
- RA Query: Π_{id}(student)
- Calculus:

$$\{ y \mid \exists x \in \text{student} \, (x[\text{id}] = y[\text{id}]) \}$$

Example #3

- student(STUDENT), enrol(ENROL)
- STUDENT = {id,name,major,GPA}
- ENROL = {id,course,grade}
- RA Query: $\Pi_{name,grade}(student \bowtie enrol)$
- Calculus:

$$\{ x \mid \exists y \in student \ \exists z \in enrol$$
$$(x[name] = y[name] \wedge$$
$$x[grade] = z[grade] \wedge$$
$$y[id] = z[id]) \}$$