CS411
Database Systems

06: SQL

Kazuhiro Minami

Modifications

Three ways to change the instance of a database:

INSERT new tuples

DELETE existing tuples

UPDATE existing tuples

Add to **Likes(drinker, beer)** the fact that Sally likes Bud.

```
INSERT INTO Likes
VALUES ('Sally', 'Bud');
        or
INSERT INTO Likes(beer, drinker)
VALUES ('Bud', 'Sally');
```

Which is better? The second one. Why? It's harder to screw up.

```
INSERT INTO Likes(beer)
VALUES ('Bud');
```

What will this do? Add a tuple with NULLs to the drinker attribute.

You can insert lots of tuples at once, using a subquery

```
INSERT INTO relation
( subquery );
```

Find Sally’s potential friends: drinkers who go to some bar that Sally goes to.

Frequents(drinker, bar)

The other drinker

```
INSERT INTO SallysPotentialFriends
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = ‘Sally’ AND
   d2.drinker <> ‘Sally’ AND
   d1.bar = d2.bar
);
```

Pairs of Drinker tuples where the first is about Sally, the second is about someone else, and the bars are the same.

Exercise: Avoid inserting duplicate tuples

Our cs411 bookstore purchased another online book store Amazon.
Insert tuples in Amazon_Books(<u>isbn</u>, title) into CS411_Books(<u>isbn</u>, title) without making duplicate tuples.

You can delete all tuples that satisfy a WHERE clause.

Likes(drinker, beer)

Delete the fact that Sally likes Bud:

```
DELETE FROM Likes
WHERE drinker = ‘Sally’ AND
      er = ‘Bud’;
```

Must specify conditions on tuples to be deleted

Delete all beers made by manufacturers who make more than one beer.

Beers(name, manf)

```
DELETE FROM Beers b
WHERE EXISTS (
  SELECT name FROM Beers
  WHERE manf = b.manf AND
        name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

You should be able to think of a few other ways to write this same query.

What happens when we run that query on this relation?

Beer	Manf
~~Bud~~	~~Anheuser-Busch~~
Bud Lite	Anheuser-Busch

Subquery is nonempty, because of the Bud Lite tuple.

Is subquery empty now? Do we delete this?

Tuples are marked for deletion, then deleted.

Beer	Manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch

Subquery is nonempty.

Subquery is nonempty.

You can change the values of selected tuples.

```
UPDATE relation
SET        attribute assignments
WHERE   condition;
```

Change drinker Fred's phone number to 555-1212.

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```

You can update several tuples at once.

Employees (EmpName, Department, HourlySalary)

Raise the minimum wage to $10/hour.

```
UPDATE   Employees
SET        HourlySalary = 10.00
WHERE     hourlySalary < 10.00;
```

Let's give everyone a 10% raise.

Employees (EmpName, Department, HourlySalary)

```
UPDATE   Employees
SET      HourlySalary = 1.1 * hourlySalary;
```

Setting Up and Changing the Database Schema:

How to declare relations, keys, and a few other things

It's very easy to create and drop relations

CREATE TABLE *relationName* (
attributeName$_1$ *type*$_1$,
…
attributeName$_K$ *type*$_K$);

DROP TABLE *relationName*;

INT or INTEGER
REAL or FLOAT
DATE 'yyyy-mm-dd'
TIME 'hh:mm:ss'
CHAR(*n*) = fixed-length string of *n* characters.
VARCHAR(*n*) = variable-length string of up to *n* characters.

```
CREATE TABLE Sells (
        bar         CHAR(20),
        beer        VARCHAR(20),
        price               REAL
);

DROP TABLE Sells;
```

You can declare your keys, using PRIMARY KEY or UNIQUE

Then the DBMS will enforce that if two tuples agree on the attribute(s) in the key, then they must agree on all of their attributes.

```
CREATE TABLE Beers (
        name        CHAR(20) PRIMARY KEY,
        manf        CHAR(20)
);
```

Sometimes a key has more than one attribute, and then we use a different syntax.

Sells (<u>bar</u>, <u>beer</u>, price)

```
CREATE TABLE Sells (
        bar         CHAR(20),
        beer        VARCHAR(20),
        price                   REAL,
        PRIMARY KEY (bar, beer)
);
```

The PRIMARY KEY is your ***favorite*** key; other keys are just UNIQUE

netID versus studentNumber : which to pick as primary key?

Some DBMSs may assume that you will usually look up tuples by their primary key, and do special things automatically to make lookups fast on those attributes.

Why label other keys as UNIQUE? The DBMS will automatically generate an error if someone does an update that violates the uniqueness constraint.

No NULLs are allowed in PRIMARY KEYs.

UNIQUE attributes can have NULLs,

and there may be several tuples with NULLs for their UNIQUE attributes.

This Employee table instance is legal, even though the two tuples (almost) agree on Name.

Primary key Unique

SSN	Name	Department	Project	Salary
123456789	NULL	NULL	Optimizer	10000
234567891	NULL	NULL	Optimizer	20000

You can prevent an attribute from ever being NULL

```
CREATE TABLE Drinkers (
    name CHAR(30) PRIMARY KEY,
    addr CHAR(50) NOT NULL,
    phone CHAR(16) DEFAULT 'Unlisted'
);
```

Saves time

You can add and drop attributes (but don't do it too often)

ALTER TABLE *relationName*
ADD *attributeName typeInfo*;

ALTER TABLE *relationName*
DROP *attributeName*;

```
ALTER TABLE Bars ADD
phone CHAR(16) DEFAULT 'unlisted';
ALTER TABLE Bars DROP phone;
```

Views

Views are like the temporary relations we declared and reused so often in relational algebra to make things easier

CREATE VIEW *viewName* AS *query*;

Then use *viewName* in queries like a “real” relation, **even though its data isn’t actually stored.** It’s a virtual relation.

A relation whose instance is really stored in the database is a *base table*.

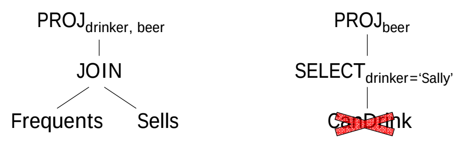
I want a view that contains just my CS411 students

Enrollments(netID, course, semester, year, grade)

```
CREATE VIEW cs411students AS
SELECT netID, grade
FROM    Enrollments
WHERE course = ‘CS411’ AND year = 2008
   AND semester = ‘fall’;
```

The DBMS replaces the view name by its definition at run time, essentially

1. Rewrite the query and the view definition in relational algebra (almost).
2. In the query, replace the view name by its definition.



Expression trees

Sometimes it makes sense to modify the tuples in a view.

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project, department
   FROM  Employee
   WHERE department = ‘Development’;
```

```
INSERT INTO  Developers
VALUES(‘Joe’, ‘Optimizer’, ‘Development’);
```

Result:

```
INSERT INTO  Employee
VALUES(NULL, ‘Joe’, Development, ‘Optimizer’, NULL);
```

Warning: as we will see later, such insertions are prohibited if the null fields are part of the primary key.

Other times, the modifications make no sense.

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = 'Development';
```

```
INSERT INTO  Developers
VALUES('Joe', 'Optimizer');
```

Result:

```
INSERT INTO  Employee
VALUES(NULL, 'Joe', NULL, 'Optimizer', NULL);
```

Joe is NOT in the view, and your users are VERY confused!

Why isn't Joe in the view?

SSN	Name	Department	Project	Salary
NULL	Joe	NULL	Optimizer	NULL

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = 'Development';
```

Joe doesn't satisfy this WHERE clause---it evaluates to MAYBE.

Of course your users don't understand this inexplicable behavior. Fortunately, later on we'll see ways to help hide these strange things from them.

Non-Updatable Views

Person(name, address), Purchase(buyer, product, store)

```
CREATE VIEW  Char

      SELECT  name, add   , product, store
      FROM     Person, Purchase
      WHERE   Person.address = 'Champaign'   AND
               Person.name = Purchase.buyer
```

Multiple relations

How can we add the following tuple to the view?

('Joe', 'Champaign', 'Nike shoes', 'Nine West')

We need to add "Joe" to Person first. One copy ? More copies ?

Non-Updatable Views

INSERT Champaign-view
VALUES ('Joe', 'Champaign', 'Nike shoes', 'Nine West')
INSERT Champaign-view
VALUES ('Joe', 'Champaign', 'U of I T-shirt', 'Nine West')

Person

name	address
Joe	Champaign
Joe	Champaign

Purchase

buyer	product	store
Joe	Nike shoes	Nine West
Joe	U of I T-shirt	WalMart

Similar troubles occur with modifications to UNION views, INTERSECTs, GROUP BYs, etc.

Often the application writer knows which interpretation makes sense, but the DBMS doesn’t automatically know.

Best solution: let the application writer catch these modification requests and rewrite them in a form that makes sense for the application.