More SQL

Database Modification

Defining a Database Schema

Views

Database Modifications

- A *modification* command does not return a result (as a query does), but changes the database in some way.
- Three kinds of modifications:

1. *Insert* a tuple or tuples.
2. *Delete* a tuple or tuples.
3. *Update* the value(s) of an existing tuple or tuples.

Insertion

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);

- Example: add to Likes(drinker, beer) the fact that Sally likes Bud.

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

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes.
- Two reasons to do so:
  1. We forget the standard order of attributes for the relation.
  2. We don’t have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

Example: Specifying Attributes

- Another way to add the fact that Sally likes Bud to Likes(drinker, beer):

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

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

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

Example: Insert a Subquery

- Using Frequents(drinker, bar), enter into the new relation PotBuddies(name) all of Sally's "potential buddies," i.e., those drinkers who frequent at least one bar that Sally also frequents.

The other drinker

Solution

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

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

Deletion

- To delete tuples satisfying a condition from some relation:

```
    DELETE FROM <relation>
    WHERE <condition>;
```

Example: Deletion

- Delete from Likes(drinker, beer) the fact that Sally likes Bud:

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

Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

Example: Delete Many Tuples

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
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.

Semantics of Deletion --- (1)

- Suppose Anheuser-Busch makes only Bud and Bud Lite.
- Suppose we come to the tuple *b* for Bud first.
- The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- Now, when *b* is the tuple for Bud Lite, do we delete that tuple too?

Semantics of Deletion --- (2)

- Answer: we *do* delete Bud Lite as well.
- The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied.
  2. Delete the marked tuples.

Updates

- To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;
```

Example: Update

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

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

Example: Update Several Tuples

- Make $4 the maximum price for beer:

```
UPDATE Sells
SET price = 4.00
WHERE price > 4.00;
```

Defining a Database Schema

- A *database schema* comprises declarations for the relations ("tables") of the database.
- Several other kinds of elements also may appear in the database schema, including views, indexes, and triggers, which we'll introduce later.

Creating (Declaring) a Relation

- Simplest form is:

```
CREATE TABLE <name> (
        <list of elements>
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

Elements of Table Declarations

- Most basic element: an attribute and its type.
- The most common types are:
  - INT or INTEGER (synonyms).
  - REAL or FLOAT (synonyms).
  - CHAR(*n*) = fixed-length string of *n* characters.
  - VARCHAR(*n*) = variable-length string of up to *n* characters.

Example: Create Table

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

Dates and Times

- DATE and TIME are types in SQL.
- The form of a date value is:

  DATE ’yyyy-mm-dd’

  - Example: `DATE ’2004-09-30’` for Sept. 30, 2004.

Times as Values

- The form of a time value is:

  TIME 'hh:mm:ss'

with an optional decimal point and fractions of a second following.

  - Example: `TIME '15:30:02.5'` = two and a half seconds after 3:30PM.

Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE.
- Either says the attribute(s) so declared functionally determine all the attributes of the relation schema.
- There are a few distinctions to be mentioned later.

Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

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

Declaring Multiattribute Keys

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement.
- This form is essential if the key consists of more than one attribute.
  - May be used even for one-attribute keys.

Example: Multiattribute Key

- The bar and beer together are the key for Sells:

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

PRIMARY KEY Versus UNIQUE

- The SQL standard allows DBMS implementers to make their own distinctions between PRIMARY KEY and UNIQUE.
  - Example: some DBMS might automatically create an *index* (data structure to speed search) in response to PRIMARY KEY, but not UNIQUE.

Required Distinctions

- However, standard SQL requires these distinctions:
  1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
  2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

Some Other Declarations for Attributes

1. NOT NULL means that the value for this attribute may never be NULL.
2. DEFAULT <value> says that if there is no specific value known for this attribute’s component in some tuple, use the stated <value>.

Example: Default Values

```
CREATE TABLE Drinkers (
   name CHAR(30) PRIMARY KEY,
   addr CHAR(50)
        DEFAULT ’123 Sesame St.’,
   phone CHAR(16)
);
```

Effect of Defaults --- (1)

- Suppose we insert the fact that Sally is a drinker, but we know neither her address nor her phone.
- An INSERT with a partial list of attributes makes the insertion possible:

```
    INSERT INTO Drinkers(name)
    VALUES('Sally');
```

Effect of Defaults --- (2)

- But what tuple appears in Drinkers?

name	**addr**	**phone**
Sally	123 Sesame St	NULL

- If we had declared phone NOT NULL, this insertion would have been rejected.

Adding Attributes

- We may add a new attribute (“column”) to a relation schema by:

  ALTER TABLE <name> ADD
      <attribute declaration>;

- Example:

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

Deleting Attributes

- Remove an attribute from a relation schema by:

```
    ALTER TABLE <name>
        DROP <attribute>;
```

- Example: we don’t really need the license attribute for bars:

```
ALTER TABLE Bars DROP license;
```

Views

- A *view* is a "virtual table" = a relation defined in terms of the contents of other tables and views.
- Declare by:

```
CREATE VIEW <name> AS <query>;
```

- Antonym: a relation whose value is really stored in the database is called a *base table*.

Example: View Definition

- CanDrink(drinker, beer) is a view “containing” the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
   SELECT drinker, beer
   FROM Frequents, Sells
   WHERE Frequents.bar = Sells.bar;
```

Example: Accessing a View

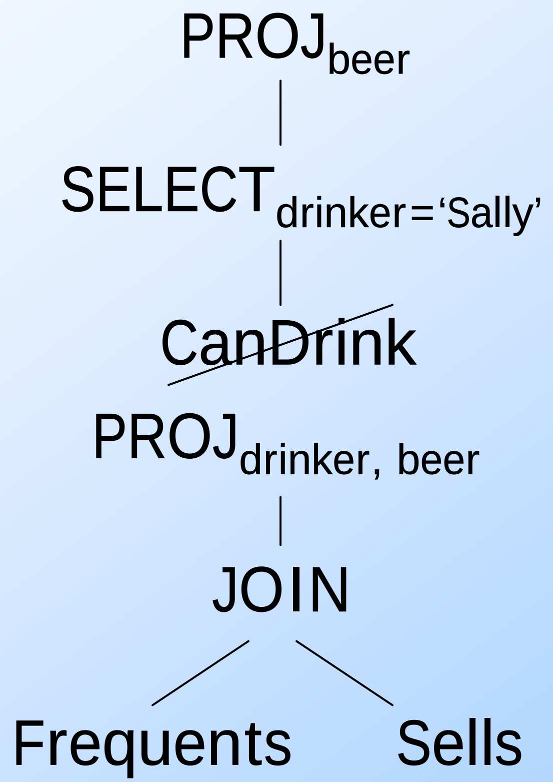
- Query a view as if it were a base table.
  - Also: a limited ability to modify views if it makes sense as a modification of one underlying base table.
- Example query:

```
SELECT beer FROM CanDrink
WHERE drinker = ’Sally’;
```

What Happens When a View Is Used?

- The DBMS starts by interpreting the query as if the view were a base table.
  - Typical DBMS turns the query into something like relational algebra.
- The definitions of any views used by the query are also replaced by their algebraic equivalents, and “spliced into” the expression tree for the query.

Example: View Expansion



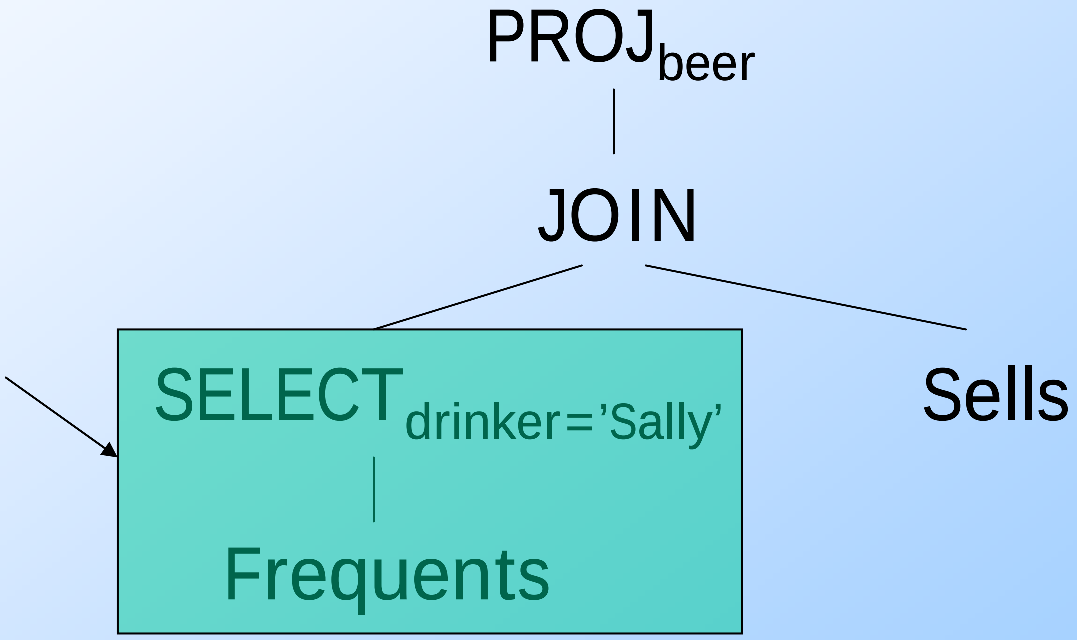
DMBS Optimization

- It is interesting to observe that the typical DBMS will then “optimize” the query by transforming the algebraic expression to one that can be executed faster.
- Key optimizations:

1. Push selections down the tree.
2. Eliminate unnecessary projections.

Example: Optimization

Notice how most tuples are eliminated from Frequents before the expensive join.



$PROJ_{beer}$
JOIN
$SELECT_{drinker='Sally'}$
Frequents
Sells