The Relational Model of Data and DataBase Modification

Jeffrey D. Ullman

Outline

- The Basics of the Relational Model
  - Relation, Attribute, Tuple
  - Schema VS Instance
- Database Schemas in SQL
- Primary Key and Foreign Key Constraints
- Views
- DataBase Modification

What is a Data Model?

1. Mathematical representation of data.

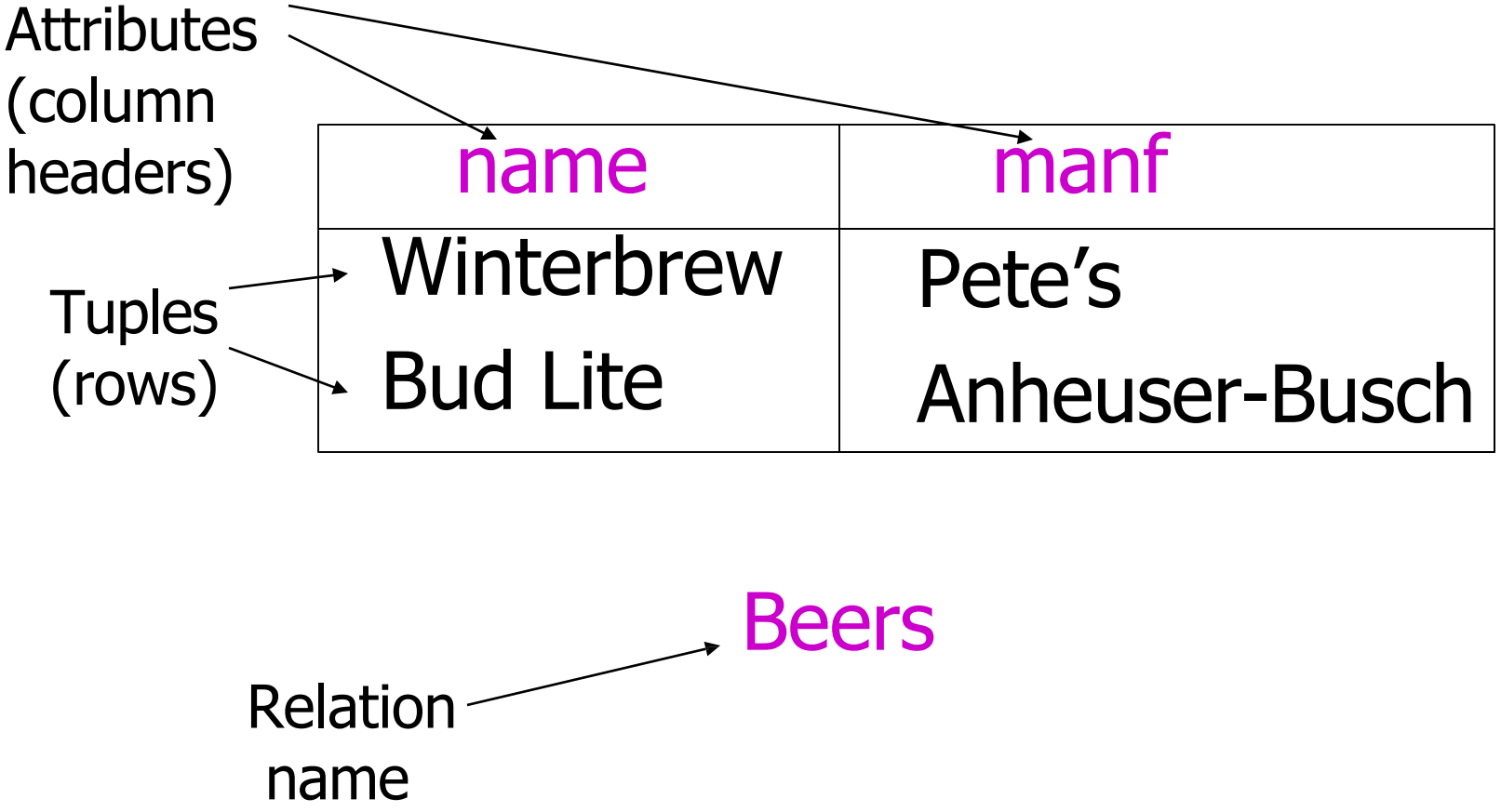
   Examples:

   - relational model = tables
   - semistructured model = trees/graphs.

2. Operations on data.
3. Constraints.

Let's examine the relational model.

A Relation is a Table



Why Relations?

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, the most important database language today.

Some Definitions

- *Relation schema* = relation name and attribute list.
  - Optionally: types of attributes.
  - Example: Beers(name, manf) or
    Beers(name: string, manf: string)
- *Relation instance* = set of tuples that the relation currently holds.
  - It changes over time.

Some Definitions (2)

- *Database* = collection of relations.
- *Database schema* = set of all relation schemas in the database.
- *Database instance* = set of all relation instances in the database.

Our Running Example

Database schema:

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Outline

- The Basics of the Relational Model
- Database Schemas in SQL
  - Table Declaration
  - Data Types
  - The NULL Value
  - Table Modification
- Primary Key and Foreign Key Constraints
- Views
- DataBase Modification

Database Schemas in SQL

- SQL is primarily a query language, for getting information from a database. This is the data-manipulation language (DML) component of SQL.

- But SQL also includes a *data-definition* component (DDL) for describing database schemas.

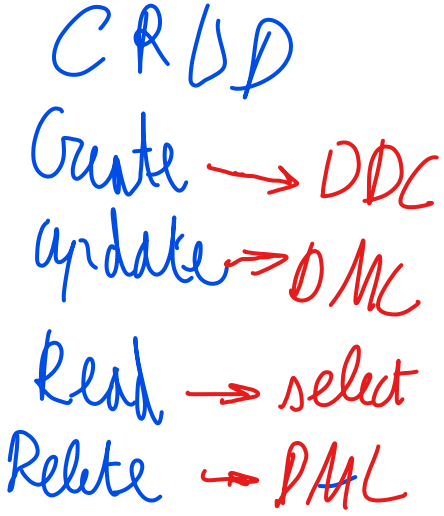
Creating (Declaring) a Relation

- 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).
  - DECIMAL(n, p) real numbers in base 10.
  - 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   DECIMAL(4,2)
);
```

SQL Values

- Integers and reals are represented as you would expect.
- Strings are too, except they require *single* quotes.
  - Two single quotes = real quote, e.g., `'Joe''s Bar'`.

Dates and Times

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

  DATE 'yyyy-mm-dd'

  - Example: `DATE '2007-09-30'` for Sept. 30, 2007.

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.

More Data Types

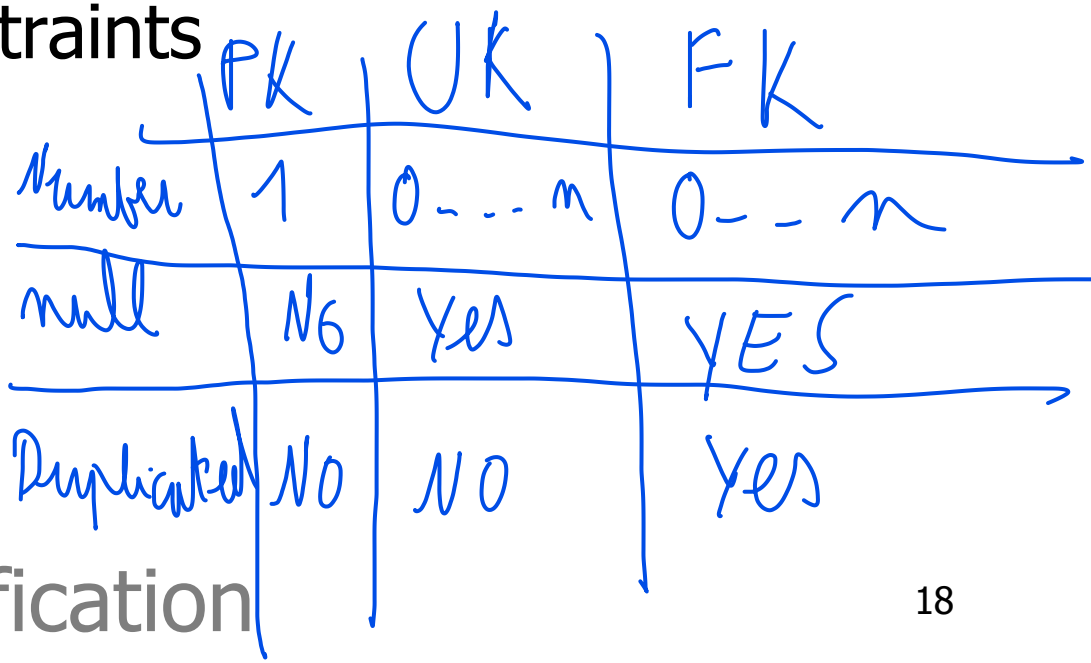
- boolean
- BLOB (Binary Large OBject): typically the content of some file (song, picture, movie, etc.)
- CLOB (Character Large OBject): typically a character file (letter, contract, etc.)

NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components.
- Meaning depends on context. Two common cases:
  - *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
  - *Inapplicable* : e.g., the value of attribute spouse for an unmarried person.

Outline

- The Basics of the Relational Model
- Database Schemas in SQL
- Primary Key and Foreign Key Constraints
  - Database Constraints
  - Primary Key
  - Unique Key
  - Foreign Key
- Views
- Database Modification



Database Constraints

- A *constraint* is a relationship among data elements that the DBMS is required to enforce.
- When a database operation violates some constraint, the DBMS usually *rejects* it.

Database Constraints (2)

- There are several kinds of constraints:
  - Keys
  - Foreign-keys, or referential-integrity
  - Later: value-based, tuple-based constraints and assertions

Key Constraint

- *Key* = set of attributes of a relation so that no two tuples may agree in all the attribute(s) of the set.
- There may be several keys for a given relation. We pick one as the *Primary Key*.
- Rule: Attributes of the primary key may not be NULL.

Declaring Keys

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

Example: Primary Key

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Underline denotes primary key attributes.

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:

- There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
- 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.

Foreign Key Constraint

- *Foreign key* = set of attributes of one relation whose values must appear *together* in certain attributes of another relation.
- There may be several foreign keys in a given relation. They usually reference several other relations.
- Rule: Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Example: Foreign Key

- In Sells(bar, beer, price), we might expect that a *beer* value also appears in *Beers.name*.
- Thus, we define beer as a foreign key that references Beers.name.
- The same applies to attribute *bar* and *Bars.name*: bar is also a foreign key.
- Note: (bar, beer) is the primary key of Sells.

Example: Foreign Key (2)

Likes(#drinker, #beer)

drinker references Drinker.name; beer references Beers.name.

Sells(#bar, #beer, price)

bar references Bars.name; beer references Beers.name.

Frequents(#drinker, #bar)

drinker references Drinker.name; bar references Bars.name.

# denotes foreign key attributes

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)PRIMARY KEY,
    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)
);
```

Foreign Keys

- Values appearing in attributes of one relation must appear together in certain attributes of another relation.
- Example: in Sells(bar, beer, price), we might expect that a beer value also appears in Beers.name .

Expressing Foreign Keys

- Use keyword REFERENCES, either:
  1. After an attribute (for one-attribute keys).
  2. As an element of the schema:

  FOREIGN KEY (<list of attributes>)

  REFERENCES <relation> (<attributes>)
- Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Example: With Attribute

```
CREATE TABLE Beers (
  name    CHAR(20) PRIMARY KEY,
  manf    CHAR(20) );
CREATE TABLE Sells (
  bar     CHAR(20),
  beer    CHAR(20) REFERENCES Beers(name),
  price   REAL );
```

POSSIBLE STATEMENT SI M FOR MAT

Example: As Schema Element

```
CREATE TABLE Beers (
  name    CHAR(20) PRIMARY KEY,
  manf    CHAR(20) );
CREATE TABLE Sells (
  bar     CHAR(20),
  beer    CHAR(20),
  price   REAL,
  FOREIGN KEY(beer) REFERENCES
     Beers(name));
```

Other Declarations for Attributes

Two other declarations we can make for an attribute are:

- NOT NULL means that the value for this attribute may never be NULL.
- 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)
);
```

Adding Attributes

We may change a relation schema by adding a new attribute ("column") by:

```
ALTER TABLE <name> ADD
<attribute declaration>;
```

Example:

```
ALTER TABLE Bars
ADD phoneCHAR(16)DEFAULT'unlisted';
```

Deleting Attributes

Remove an attribute from a relation schema by:

DROP supprime

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

DELETE vide

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

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

Outline

- The Basics of the Relational Model
- Database Schemas in SQL
- Primary Key and Foreign Key Constraints
- Views
- Database Modifications

Views

A view is a "virtual table," a relation that is defined in terms of the contents of other tables and views.

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

In contrast, a relation whose value is really stored in the database is called a *base table*.

Example: View Definition

Can Drink(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

You may query a view as if it were a base table.

There is a limited ability to modify views if the modification makes sense as a modification of the underlying base table.

Example:

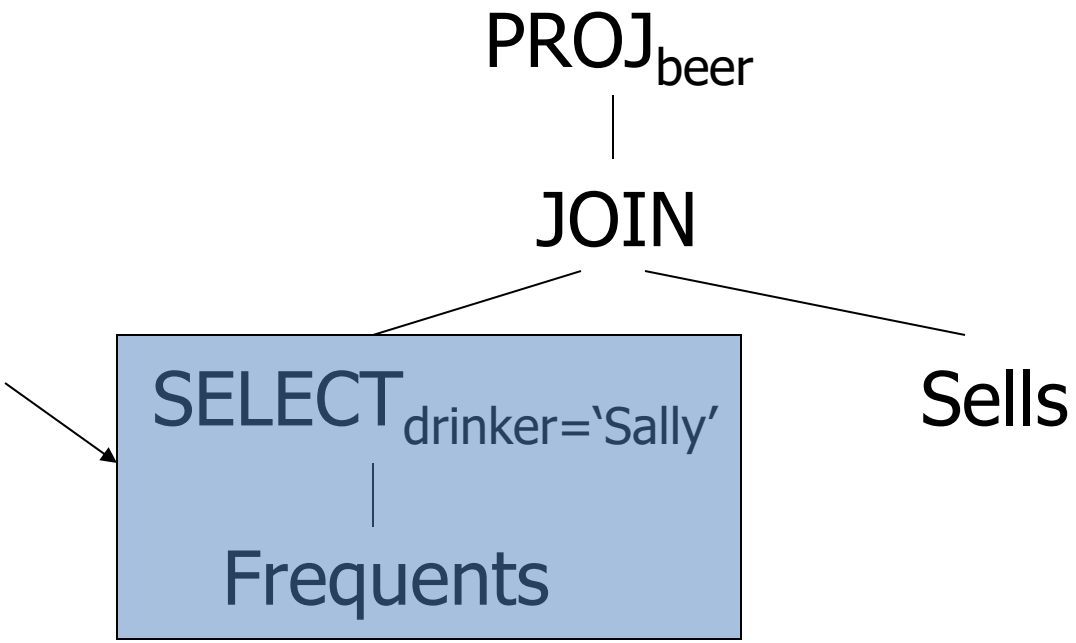
```
SELECT beer FROM CanDrink
WHERE drinker = 'Sally';
```

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 queries defining any views used by the query are also replaced by their algebraic equivalents, and "spliced into" the expression tree for the query.

Example

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



Outline

- The Basics of the Relational Model
- Database Schemas in SQL
- Primary Key and Foreign Key Constraints
- Views
- Database Modification
  - Insert
  - Delete
  - Update

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. You forgot the 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');
```

Adding Default Values

- In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.
- When an inserted tuple has no value for that attribute, the default will be used.

Example: Default Values

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

Example: Default Values

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

Resulting tuple:

name	address	phone
Sally	123 Sesame St	NULL

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.

Solution

The other drinker

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 Some 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;
```