THE LECTURE 5

DATABASE MODIFICATIONS

DB MODIFICATIONS

- *Modification* = insert + delete + update.

Insertion of a Tuple

`INSERT INTO` relation `VALUES` (list of values).

- Inserts the tuple = list of values, associating values with attributes in the order the attributes were declared.
  - Forget the order? List the attributes as arguments of the relation.

Example

```
Likes(consumer, apple)
```

Insert the fact that Sally likes Bud.

```
INSERT INTO Likes(consumer, apple)
VALUES('Sally', 'Green');
```

INSERTION OF THE RESULT OF A QUERY

`INSERT INTO` relation (subquery).

Example

```
Frequents(consumer, shop)
```

```
CREATE TABLE PotBuddies(
   name char(30)
);
```

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

DELETION

`DELETE FROM` relation `WHERE` condition.

- Deletes all tuples satisfying the condition from the named relation.

Example

Sally no longer likes Bud.

`Likes(consumer, apple)`

```
DELETE FROM Likes
WHERE consumer = 'Sally' AND
    apple = ‘Green';
```

Example

Make the `Likes` relation empty.

```
DELETE FROM Likes;
```

EXAMPLE

- Delete all apples for which there is another apple by the same manufacturer.

```
Apples(name, manf)

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

- Note alias for relation from which deletion occurs.

UPDATES

`UPDATE` relation `SET` list of assignments `WHERE` condition.

Example

Drinker Fred's phone number is 555-1212.

```
Consumers(name, addr, phone)

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

Example

Make $4 the maximum price for apple.

- Updates many tuples at once.

```
Sells(shop, apple, price)

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

DEFINING A DATABASE SCHEMA

`CREATE TABLE` name (list of elements).

- Principal elements are attributes and their types, but key declarations and constraints also appear.
- Similar `CREATE` *X* commands for other schema elements *X*: views, indexes, assertions, triggers.
- "`DROP` *X* name" deletes the created element of kind *X* with that name.

Example

```
    CREATE TABLE Sells (
         shop CHAR(20),
         name VARCHAR(20),
         price REAL
    );

    DROP TABLE Sells;
```

TYPES

- `INT` or `INTEGER`.
- `REAL` or `FLOAT`.
- `CHAR(`*n*`)` = fixed length character string, padded with “pad characters.”
- `VARCHAR(`*n*`)` = variable-length strings up to *n* characters.
  - Oracle uses `VARCHAR2(`*n*`)` as well. PostgreSQL uses `VARCHAR` and does not support `VARCHAR2`.

TYPES

- `NUMERIC(`*precision, decimal*`)` is a number with *precision* digits with the decimal point *decimal* digits from the right. `NUMERIC(10,2)` can store ±99,999,999.99
- `DATE`. SQL form is `DATE 'yyyy-mm-dd'`
  - PostgreSQL follows the standard. Oracle uses a different format.
- `TIME`. Form is `TIME 'hh:mm:ss[.ss…]'` in SQL.
- `DATETIME` or `TIMESTAMP`. Form is `TIMESTAMP 'yyyy-mm-dd hh:mm:ss[.ss…]'` in SQL.
- `INTERVAL`. Form is `INTERVAL 'n` *period*' in PostgreSQL. *Period* is `month`, `days`, `year`, etc.

DECLARING KEYS

Use `PRIMARY KEY` or `UNIQUE`.

- But only one primary key, many `UNIQUE`s allowed.
- SQL permits implementations to create an *index* (data structure to speed access given a key value) in response to `PRIMARY KEY` only.
  - But PostgreSQL and Oracle create indexes for both.
- SQL does not allow nulls in primary key, but allows them in "unique" columns (which may have two or more nulls, but not repeated non-null values).

DECLARING KEYS

Two places to declare:

- After an attribute's type, if the attribute is a key by itself.
- As a separate element.
  - Essential if key is >1 attribute.

EXAMPLE

```
CREATE TABLE Sells (
    shop CHAR(20),
    apple VARCHAR(20),
    price REAL,
    PRIMARY KEY(shop,apple)
);
```

EXAMPLE

```
CREATE TABLE Sells (
    shop CHAR(20),
    apple VARCHAR(20),
    price REAL,
    UNIQUE(shop,apple)
);
```

is different than:

```
CREATE TABLE Sells (
     shop CHAR(20) UNIQUE,
     apple VARCHAR(20) UNIQUE,
     price REAL
 );
```

OTHER PROPERTIES YOU CAN GIVE TO ATTRIBUTES

- `NOT NULL` = every tuple must have a real value for this attribute.
- `DEFAULT` value = a value to use whenever no other value of this attribute is known.

Example

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

OTHER PROPERTIES YOU CAN GIVE TO ATTRIBUTES

```
INSERT INTO Consumers(name)
VALUES('Sally')
```

results in the following tuple:

name	addr	phone
Sally	123 Sesame St.	NULL

- Primary key is by default not NULL.
- This insert is legal.
  - OK to list a subset of the attributes and values for only this subset.
- But if we had declared

```
    phone CHAR(16) NOT NULL
```

then the insertion could not be made.

INTERESTING DEFAULTS

- `DEFAULT CURRENT_TIMESTAMP`
- `SEQUENCE`

```
CREATE SEQUENCE customer_seq;
CREATE TABLE Customer (
     customerID INTEGER
       DEFAULT nextval('customer_seq'),
     name VARCHAR(30)
);
```

CHANGING COLUMNS

Add an attribute of relation *R* with

`ALTER TABLE` *R* `ADD` <column declaration>`;`

Example

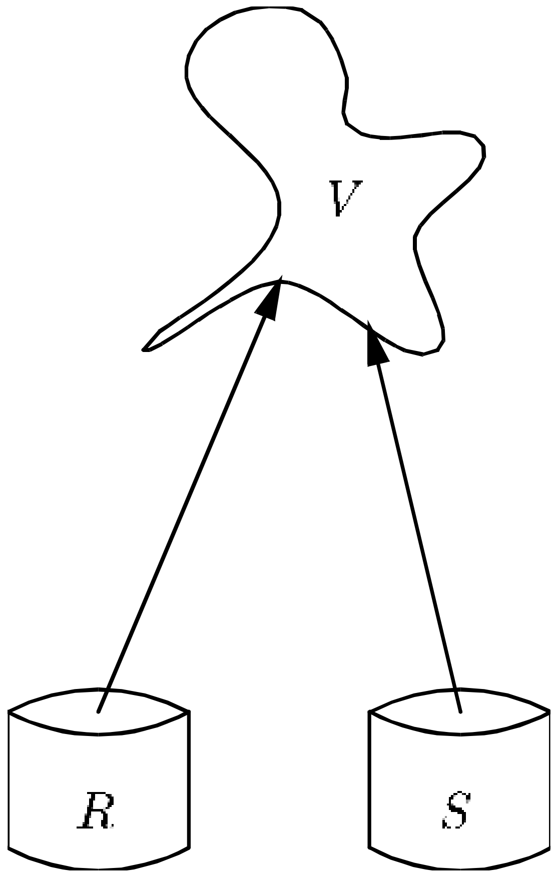
```
ALTER TABLE Shops ADD phone CHAR(16)
    DEFAULT 'unlisted';
```

- Columns may also be dropped.

```
ALTER TABLE Shops DROP license;
```

VIEWS

An expression that describes
a table without creating it.



- View definition form is:

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

EXAMPLE

The view `CanConsume` is the set of consumer-apple pairs such that the consumer frequents at least one apple that serves the apple.

```
CREATE VIEW CanConsume AS
     SELECT consumer, apple
     FROM Frequents, Sells
     WHERE Frequents.apple = Sells.apple;
```

Querying Views

Treat the view as if it were a materialized relation.

Example

```
SELECT apple
FROM CanConsume
WHERE consumer = ‘Sally’;
```