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 (
    bar CHAR(20),
    beer VARCHAR(20),
    price REAL
);

DROP TABLE Sells;
```

Types

1. `INT` or `INTEGER`.

2. `REAL` or `FLOAT`.

3. `CHAR(`n`)` = fixed length character string, padded with "pad characters."

4. `VARCHAR(`n`)` = variable-length strings up to n characters.

    - Oracle uses `VARCHAR2(`n`)` as well. Difference: storage for `VARCHAR2` is truly varying length; `VARCHAR` uses fixed array with endmarker.

    - `VARCHAR` in Oracle is "deprecated" (they may discontinue it in the future), so they suggest you always use `VARCHAR2`.

5. Dates. SQL form is `DATE 'yyyy-mm-dd'`
    - Oracle uses a different format — to be explained.
6. Times. Form is `TIME 'hh:mm:ss[.ss...]'` in SQL.
7. In Oracle: `NUMBER` is either integer or floating point as appropriate.

Oracle Default Dates (Used at Stanford)

Format `'dd-mon-yy'`

- Behind the scenes (as stored in the relation), the value of a date field has as much precision as the computer allows.

Example

```
CREATE TABLE Days (
    d DATE
);

INSERT INTO Days
VALUES('06-nov-97');
```

- Oracle function `to_date` converts a specified format into default format.

```
INSERT INTO Days
VALUES(to_date('2000-01-01',
        'yyyy-mm-dd'));
```

Declaring Keys

Use `PRIMARY KEY` or `UNIQUE`.

- Oracle treats these as synonyms.
- But only one primary key, many "uniques" allowed.
- SQL permits implementations to create an *index* (data structure to speed access given a key value) in response to `PRIMARY KEY` only.
    - But Oracle creates 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 nonnull values).

Declaring Keys

Two places to declare:

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

Example

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

- On the Stanford Oracle system for this class, there is a separate data area on a separate disk for indexes.
  - ✦ Speeds access — two heads are better than one.
  - ✦ Thus, you should follow any implicit index-creating statement like "primary key," by:

```
USING INDEX TABLESPACE indx
```

Example

```
CREATE TABLE Beers (
    name CHAR(20) UNIQUE
        USING INDEX TABLESPACE indx,
    manf CHAR(20)
);
```

Other Properties You Can Give to Attributes

1. `NOT NULL` = every tuple must have a real value for this attribute.

2. `DEFAULT` value = a value to use whenever no other value of this attribute is known.

Example

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

```
INSERT INTO Drinkers(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.

Changing Columns

Add an attribute of relation R with

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

Example

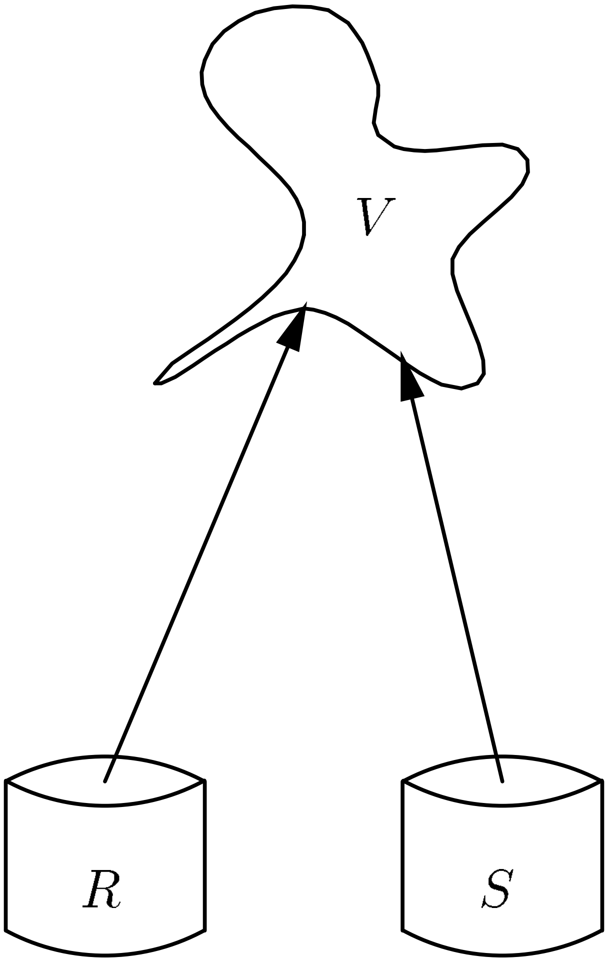
```
ALTER TABLE Bars ADD phone CHAR(16)
          DEFAULT 'unlisted';
```

- Columns may also be dropped.

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

Views

An expression that describes a table without creating it.



- View definition form is:

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

Example

The view `CanDrink` is the set of 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;
```

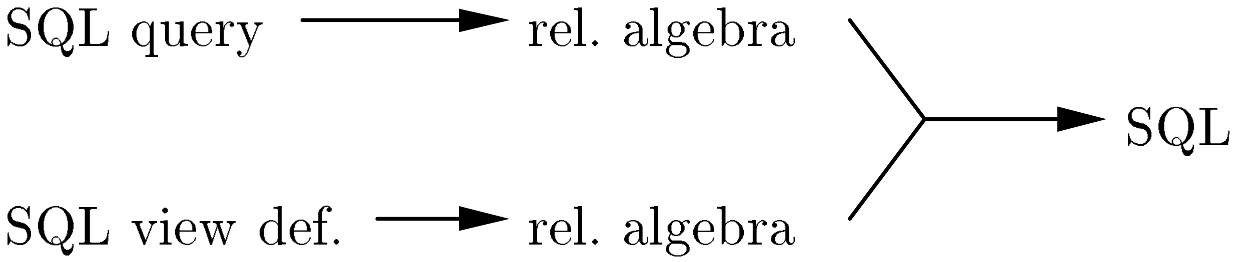
Querying Views

Treat the view as if it were a materialized relation.

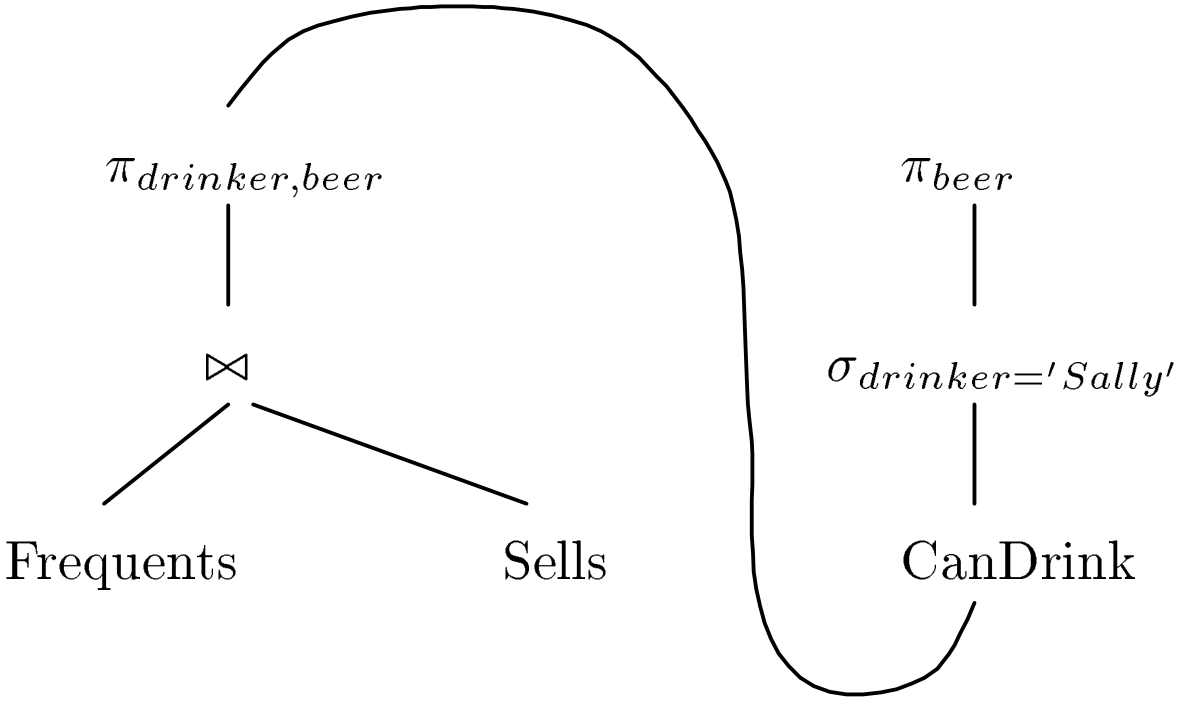
Example

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

Semantics of View Use

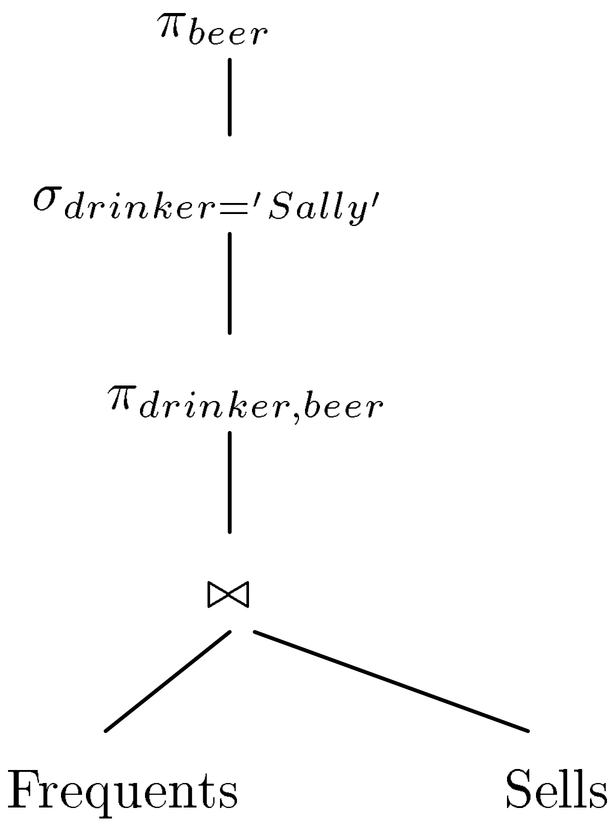


Example



`CanDrink` `Query`

Compose


π_{beer}
$\sigma_{drinker='Sally'}$
$\pi_{drinker,beer}$
$\bowtie$
Frequents
Sells

Optimize Query

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