5.8 View Definitions

Relations that are defined with a `CREATE TABLE` statement actually exist in the database. That is, an SQL system stores tables in some physical organization. They are persistent, in the sense that they can be expected to exist indefinitely and not to change unless they are explicitly told to change by an `INSERT` or one of the other modification statements we discussed in Section 5.6.

There is another class of SQL relations, called *views*, that do not exist physically. Rather, they are defined by an expression much like a query. Views, in turn, can be queried as if they existed physically, and in some cases, we can even modify views.

5.8.1 Declaring Views

The simplest form of view definition is

1. The keywords `CREATE VIEW`,
2. The name of the view,
3. The keyword `AS`, and
4. A query Q. This query is the definition of the view. Any time we query the view, SQL behaves as if Q were executed at that time and the query applied to the relation produced by Q.

That is, a simple view declaration has the form

`CREATE VIEW` <view-name> `AS` <view-definition>;

Example 5.37: Suppose we want to have a view that is a part of the

`Movie(title, year, length, inColor, studioName, producerC#)`

relation, specifically, the titles and years of the movies made by Paramount Studios. We can define this view by

```
1) CREATE VIEW ParamountMovie AS
2)     SELECT title, year
3)     FROM Movie
4)     WHERE studioName = 'Paramount';
```

First, the name of the view is `ParamountMovie`, as we see from line (1). The attributes of the view are those listed in line (2), namely `title` and `year`. The definition of the view is lines (2) through (4). □

Relations, Tables, and Views

SQL programmers tend to use the term "table" instead of "relation." The reason is that it is important to make a distinction between stored relations, which are "tables," and virtual relations, which are "views." Now that we know the distinction between a table and a view, we shall use "relation" only where either a table or view could be used. When we want to emphasize that a relation is stored, rather than a view, we shall sometimes use the term "base relation" or "base table."

There is also a third kind of relation, one that is neither a view nor stored permanently. These relations are temporary results, as might be constructed for some subquery. Temporaries will also be referred to as "relations" subsequently.

5.8.2 Querying Views

Relation `ParamountMovie` does not contain tuples in the usual sense. Rather, if we query `ParamountMovie`, the appropriate tuples are obtained from the base table `Movie`, so the query can be answered. As a result, we can ask the same query about `ParamountMovie` twice and get different answers, even though we appear not to have changed `ParamountMovie`, because the base table may have changed in the interim.

Example 5.38: We may query the view `ParamountMovie` just as if it were a stored table, for instance:

```
SELECT title
FROM ParamountMovie
WHERE year = 1979;
```

The definition of the view `ParamountMovie` is used to turn the query above into a new query that addresses only the base table `Movie`. We shall illustrate how to convert queries on views to queries on base tables in Section 5.8.5. However, in this simple case it is not hard to deduce what the example query about the view means. We observe that `ParamountMovie` differs from `Movie` in only two ways:

1. Only attributes `title` and `year` are produced by `ParamountMovie`.
2. The condition `studioName = 'Paramount'` is part of any `WHERE` clause about `ParamountMovie`.

Since our query wants only the `title` produced, (1) does not present a problem. For (2), we need only to introduce the condition `studioName = 'Paramount'` into the `WHERE` clause of our query. Then, we can use `Movie` in place of

ParamountMovie in the **FROM** clause, assured that the meaning of our query is preserved. Thus, the query:

```
SELECT title
FROM Movie
WHERE studioName = 'Paramount' AND year = 1979;
```

is a query about the base table **Movie** that has the same effect as our original query about the view **ParamountMovie**. Note that it is the job of the SQL system to do this translation. We show the reasoning process only to indicate what a query about a view means. □

Example 5.39: It is also possible to write queries involving both views and base tables. An example is

```
SELECT DISTINCT starName
FROM ParamountMovie, StarsIn
WHERE title = movieTitle AND year = movieYear;
```

This query asks for the name of all stars of movies made by Paramount. Note that the use of **DISTINCT** assures that stars will be listed only once, even if they appeared in several Paramount movies. □

Example 5.40: Let us consider a more complicated query used to define a view. Our goal is a relation **MovieProd** with movie titles and the names of their producers. The query defining the view involves both relation

```
Movie(title, year, length, inColor, studioName, producerC#)
```

from which we get a producer's certificate number, and the relation

```
MovieExec(name, address, cert#, netWorth)
```

where we connect the certificate to the name. We may write:

```
1) CREATE VIEW MovieProd AS
2)     SELECT title, name
3)     FROM Movie, MovieExec
4)     WHERE producerC# = cert#;
```

We can query this view as if it were a stored relation. For instance, to find the producer of *Gone With the Wind*, ask:

```
SELECT name
FROM MovieProd
WHERE title = 'Gone With the Wind';
```

As with any view, this query is treated as if it were an equivalent query over the base tables alone, such as:

```
SELECT name
FROM Movie, MovieExec
WHERE producerC# = cert# AND title = 'Gone With the Wind';
```

□

5.8.3 Renaming Attributes

Sometimes, we might prefer to give a view's attributes names of our own choosing, rather than use the names that come out of the query defining the view. We may specify the attributes of the view by listing them, surrounded by parentheses, after the name of the view in the `CREATE VIEW` statement. For instance, we could rewrite the view definition of Example 5.40 as:

```
CREATE VIEW MovieProd(movieTitle, prodName) AS
    SELECT title, name
    FROM Movie, MovieExec
    WHERE producerC# = cert#;
```

The view is the same, but its columns are headed by attributes `movieTitle` and `prodName` instead of `title` and `name`.

5.8.4 Modifying Views

In limited circumstances it is possible to execute an insertion, deletion, or update to a view. At first, this idea makes no sense at all, since the view does not exist the way a base table (stored relation) does. What could it mean, say, to insert a new tuple into a view? Where would the tuple go, and how would the database system remember that it was supposed to be in the view?

For many views, the answer is simply "you can't do that." However, for sufficiently simple views, called *updatable views*, it is possible to translate the modification of the view into an equivalent modification on a base table, and the modification can be done to the base table instead. SQL2 provides a formal definition of when modifications to a view are permitted. The SQL2 rules are complex, but roughly, they permit modifications on views that are defined by selecting (using `SELECT`, not `SELECT DISTINCT`) some attributes from one relation R (which may itself be an updatable view). Two important technical points:

- The `WHERE` clause must not not involve R in a subquery.
- The attributes in the `SELECT` clause must include enough attributes that for every tuple inserted into the view, we can fill the other attributes out with `NULL` values or the proper default and have a tuple of the base relation that will yield the inserted tuple of the view.