By Marina Barsky

Advanced features of relational databases

Lecture 10

VIEWS

Views

- A view is a "*virtual table*", a relation defined in terms of the contents of other tables and views
- Views take very little space to store - the database contains only the definition of a view, not a copy of all the data that it presents
- In contrast, a relation whose value is really stored in the database is called a *base table*

Example

```
CREATE VIEW DMovies AS
    SELECT title, year, length
    FROM Movie
    WHERE studioName = 'Disney';
```

Querying a View

- Query a view as if it were a base table.

Examples

```
SELECT title
FROM DMovies
WHERE year = 1990;

SELECT DISTINCT starName
FROM DMovies, StarsIn
WHERE DMovies.title = StarsIn.movietitle AND DMovies.year
  = StarsIn.movieyear;
```

View on more than one relation

```
CREATE VIEW  MoviesAndStars AS
        SELECT Movies.Title asa title,  Movies.year as year,
MovieStar.name  as star                 FROM Movies, StarsIn,
MovieStar
        WHERE Movie.title= StarsIn.movietitle
        AND Movies.year= StarsIn.movieyear
        AND MovieStar.name= StarsIn.starname;

SELECT * FROM MoviesAndStars;
```

Modifying views

- The view does not exist as a stored relation – can we modify the data in a view?
  - Where the inserted tuple should be stored?
  - Does the deletion/update affect the base table?

- In most cases the answer is: views are not modifiable

- For some simple views and some DBMSs there are updatable views

When modifications to a view are permitted

- If the view is defined by SELECT (not SELECT DISTINCT) from some relation R:
  1. R is the only relation in the FROM clause
  2. R is not in the subquery of the WHERE clause of this view
  3. The list in the SELECT clause includes enough attributes such that for every tuple inserted into the view, we can fill the other attributes out with NULL or the default, and have a tuple that will yield the valid insertion into R -
- In this case the insertion or update of the view can be applied directly to relation R

Updateable view example

THIS EXAMPLE WILL WORK WITH ORACLE AND POSTGRESQL V 9.3, we have V 9.1 which does not have automatic support for updateable views

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount';
```

Updateable view example: conditions for being updateable

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount';
```

R is the only relation in the FROM clause

Updateable view example: conditions for being updateable

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount';
```

R is the only relation in the FROM clause

R is not in the subquery of the WHERE clause of this view

Updateable view example: conditions for being updateable

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount';
```

R is the only relation in the FROM clause

R is not in the subquery of the WHERE clause of this view

View has enough attributes for the insertion into R

Updateable view example: insertion

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount';

INSERT INTO ParamountMovie
VALUES ('Star Trek', 1979);
```

This insertion will fail!

Why this insertion is not possible?

Updateable view example: failed insertion

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movie
    WHERE studioName = 'Paramount';

INSERT INTO ParamountMovie
VALUES ('Star Trek', 1979);
```

Why this insertion is not possible?

The rationale for this behavior is:

- The above insertion, were it allowed to get through, would insert a tuple with NULL for studioName in the underlying Movie table.
- However, such a tuple doesn't satisfy the condition for being in the ParamountMovie view!
- Thus, it shouldn't be allowed to get into the database through the ParamountMovie view.

Updateable view example: fixed

```
CREATE VIEW ParamountMovie2 AS
    SELECT studioName, title, year
    FROM Movie
    WHERE studioName = 'Paramount‘;
```

```
INSERT INTO ParamountMovie2
VALUES ('Paramount', 'Star Trek', 1979);
```

Now it succeeds. Why?

Deleting from an updateable view

```
DELETE FROM ParamountMovie
WHERE year=2008;
```

is translated into

```
DELETE FROM Movies
WHERE year=2008 AND studioName='Paramount';
```

Updating an updateable view

```
UPDATE ParamountMovie
SET year = 1979
WHERE title= 'Star Trek';
```

is equivalent to the base-table update

```
UPDATE Movies
SET year = 1979
WHERE title = 'Star Trek' AND
                        studioName = 'Paramount';
```

Use of updateable views

- Often used to restrict user input

- For example, you could not by mistake:

```
INSERT INTO ParamountMovie2
VALUES (‘Disney', 'Star Trek', 1979);
```

Materialized views (Oracle)

- Effectively a database table that contains the results of a query – sort of caching

- The power of materialized views comes from the fact that, once created, Oracle can automatically synchronize a materialized view's data with its source information with little or no programming effort

SQL queries with views

Find the stars who have worked for **every** studio.

```
CREATE VIEW MovieStarView AS
    SELECT title, year, studioName, starName
    FROM Movies, StarsIn
    WHERE Movies.title = StarsIn.movieTitle and Movies.Year = Starsin.MovieYear;

SELECT DISTINCT starName
FROM MovieStarView X
WHERE NOT EXISTS (
    SELECT studioName
    FROM Studio
            EXCEPT
    SELECT studioName
    FROM MovieStarView
    WHERE starName = X.starName);
```

Checks emptiness of the subquery.

Correlated subquery

- W7 exercises q 1

- W7 exercises q 2

Solution W7 q 2

Find the stars who have worked for Disney but no other studio.

```
CREATE VIEW MovieStarView AS
    SELECT title, year, studioName, starName
    FROM Movies, StarsIn
    WHERE Movies.title = StarsIn.movieTitle
AND Movies.Year = Starsin.MovieYear;


SELECT starName
FROM MovieStarView X
WHERE X.studioName='Disney' AND NOT EXISTS (
   SELECT *
   FROM MovieStar
   WHERE       starName=X.starName  AND
               studioName<>'Disney'
);
```

- W7 exercises q 3

Solution W7 q 3

Find the stars who have worked for only one studio.

```
CREATE VIEW MovieStarView AS
    SELECT title, year, studioName, starName
    FROM Movies, StarsIn
    WHERE Movies.title = StarsIn.movieTitle
AND Movies.Year = Starsin.MovieYear;


SELECT starName
FROM MovieStarView X
WHERE NOT EXISTS (
    SELECT *
    FROM MovieStarView
    WHERE          starName=X.starName  AND
                   studioName<>X.studioName
);
```

- W7 exercises q 4

Solution W 7 q 4.

For each star that has more than two movies with Paramount, find how many movies he/she has with Fox.

```
CREATE VIEW ParamountStars2 AS
    SELECT starName
    FROM MovieStarView
    WHERE studioName='Paramount'
    GROUP BY starName
    HAVING COUNT(title)>=2;

CREATE VIEW FoxStars AS
    SELECT *
    FROM MovieStarView
    WHERE studioName='Fox';

SELECT starName, COUNT(title) as countFox
FROM ParamountStars2 NATURAL LEFT OUTER JOIN FoxStars
GROUP BY starName;
```

Find the stars who have co-starred with the same star.

```
CREATE VIEW costars AS
SELECT X.starname AS star1, Y.starname AS star2
FROM StarsIn X JOIN StarsIn Y USING(title,year)
WHERE X.starname <> Y.starname;
```

costars

star1	star2
A	B
A	C
D	B

Z

star1	star2
A	B
A	C
D	B

W

star1	star2
A	B
A	C
D	B

```
SELECT Z.star1, W.star1
FROM costars Z, costars W
WHERE  Z.star2=W.star2 AND Z.star1<W.star1;
```

For each pair of co-stars give the number of movies each has starred in.

The result should be a set of (star1 star2 n1 n2) quadruples, where n1 and n2 are the number of movies that star1 and star2 have starred in, respectively. Observe that there might be stars with zero movies they have starred in.

```
CREATE VIEW starMovieCounts AS
SELECT name AS star, COUNT(title) AS moviecount
FROM Stars LEFT OUTER JOIN StarsIn ON name=starname
GROUP BY name;

SELECT C.star1, C.star2, X.moviecount, Y.moviecount
FROM costars C, starMovieCounts X, starMovieCounts Y
WHERE C.star1=X.star AND C.star2=Y.star;
```

Summary: Views

- Provide modularization abstraction for SQL queries (like a function in programming languages)
- Limit the degree of exposure of the underlying tables to the outer world
- Allow to join and simplify multiple tables into a single virtual table
- Hide the complexity of data: provide logical data independence

  In your program, retrieve data from the view: if the definition of underlining tables changes, you do not need to update your code – just re-write the view