SQL: Views

Life without views

In a DBMS, a table must by physically stored. We never have to concern ourselves about how or where it is stored, but we know the DBMS will take care of that.

Sometimes we want to define new things in terms of old things. For example, we might want to define which students are on the Dean's Honours List in a given term in terms of the table that stores grades and the table that stores information about offerings of courses. We could do this by creating a new table, and putting the correct data into it:

```
csc343h-prof=> CREATE TABLE DeansHonours (sid integer, term integer, average float);
CREATE TABLE
csc343h-prof=> INSERT INTO DeansHonours
SELECT sid, term, avg(grade)
FROM Took JOIN Offering ON Took.oid = Offering.oid
GROUP BY sid, term
HAVING avg(grade) > 80;
INSERT 0 3

csc343h-prof=> SELECT * FROM DeansHonours;
  sid  | term  |     average
-------+-------+-----------------
 98000 | 20081 |              82
 98000 | 20089 |            83.8
 99999 | 20089 | 88.5555555555556
(3 rows)
```

This works nicely, but suppose we change a student's grades.

```
csc343h-prof=> -- Add a new grade for student 98000.
csc343h-prof=> INSERT INTO Took values (98000, 32, 100);
INSERT 0 1

csc343h-prof=> -- Offering 32 took place in term 20081.
csc343h-prof=> SELECT * FROM Offering WHERE oid = 32;
 oid | cnum | dept | term  | instructor
-----+------+------+-------+------------
  32 |  216 | EEB  | 20081 | Suzuki
(1 row)
```

The average for student 98000 in term 20081 should be pulled up by the 100 that we inserted. But it is

The average for student 99999 in term 20091 should be pulled up by the 100 that we inserted. But it is not:

```
csc343h-dianeh=> SELECT * FROM DeansHonours;
  sid  | term  |     average
-------+-------+-----------------
 98000 | 20081 |              82
 98000 | 20089 |            83.8
 99999 | 20089 | 88.5555555555556
(3 rows)
```

Remember that we computed and stored table DeansHonours, and we never asked to change what's in that table. And if a student should be added or dropped from the Dean's Honours List because of a grade change, table DeansHonours will not reflect that either.

If we want to recompute the averages of Dean's Honours list students, we will see the correct results:

```
csc343h-prof=> SELECT sid, term, avg(grade)
csc343h-prof-> FROM Took JOIN Offering ON Took.oid = Offering.oid
csc343h-prof-> GROUP BY sid, term
csc343h-prof-> HAVING avg(grade) > 80;
  sid  | term  |         avg
-------+-------+--------------------
 98000 | 20081 | 85.0000000000000000
 98000 | 20089 | 83.8000000000000000
 99999 | 20089 | 88.5555555555555556
(3 rows)
```

This is because we are computing this query from the now-revised Took table.

Life with (virtual) views

Views give us an alternative way to define new things in terms of old things. When we define a view, we aren't asking SQL to compute anything; we are just asking it to remember a definition. For example, this code:

```
csc343h-prof=> CREATE VIEW DeansHonoursStudents AS
csc343h-prof-> SELECT sid, term, avg(grade)
csc343h-prof-> FROM Took JOIN Offering ON Took.oid = Offering.oid
csc343h-prof-> GROUP BY sid, term
csc343h-prof-> HAVING avg(grade) > 80;
CREATE VIEW
```

does not figure out anyone's average; it just tells SQL "When I say DeansHonoursStudents, I mean that code up there". If we want SQL to do the computation, we just have to use the view in some other query. It could be as simple a query as this:

It could be as simple a query as this:

```
csc343h-prof=> SELECT * FROM DeansHonoursStudents;
  sid  | term  |         avg
-------+-------+--------------------
 98000 | 20081 | 85.0000000000000000
 98000 | 20089 | 83.8000000000000000
 99999 | 20089 | 88.5555555555555556
(3 rows)
```

or it could be a more complex query that, say, joins this view with some existing tables or other views.

Notice what happens now if we change a student's grades.

```
csc343h-prof=>  INSERT INTO Took values (98000, 33, 100);
INSERT 0 1
csc343h-prof=> SELECT * FROM DeansHonoursStudents;
  sid  | term  |         avg
-------+-------+--------------------
 98000 | 20081 | 87.1428571428571429
 98000 | 20089 | 83.8000000000000000
 99999 | 20089 | 88.5555555555555556
(3 rows)
```

Student 98000's average in term 20081, as reported here, has gone up. This is because our query uses the DeansHonoursStudents view, which is recomputed every time it is used.

This kind of view is called a "virtual view" because, other than its definition, it is not stored. (My dictionary says that a virtual thing doesn't physically exist but is made to appear as if it exists.)

Pros and cons

Views are always update because they are recomputed every time they are used. But of course this comes at the cost of all the recomputation. Derived tables don't have that cost, but (a) they don't stay up-to-date with respect to the base tables they were derived from, and (b) they require storage space for their contents, which may be very large.

Uses for views

Views can be used to break down a large query, as we did with assignment statements in relational algebra.

Views can also be used to provide another way of looking at the same data/ For example, imagine that the university has staff who review a student's transcript to determine if they have completed all the requirements of their program. For this, they might need information pulled from multiple tables, but only

some of the columns and rows. They wouldn't need to see failing grades, for instance, since they don't contribute to program completion, and they wouldn't need to see what professor taught each course because that is not relevant. A view could assemble the necessary information and present it as if that's exactly what's in the database. We can also grant users privileges to access the view but not the base tables it came from.

The other kind of view: a materialized view

A "materialized" view is actually computed and stored at the time it is defined, and in addition, the relationship between it and its base tables is maintained at all times. Like a derived table, it doesn't have to be recomputed (unless something changes in the base tables), and like a virtual view, it is always up to date. But it is tremendously expensive to maintain the relationship between a materialized view and its base tables; every change to one of them could potentially require an update to the materialized view. Because of this, many DBMSs do not support materialized views.