DATABASE SYSTEMS

Lecture 12 Additional Notes

Views

A view is the dynamic result of one or more relational operations (queries) operating on base relations (tables) to produce another relation (virtual table).

It is a virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request. The contents of a view are defined as a query on one or more base relations.

Views are dynamic, meaning that changes made to base relations (tables) that affect view attributes are immediately reflected in the view.

Purpose of Views

The view mechanism is desirable for the following reasons:

- A view provides a powerful and flexible security mechanism by hiding parts of the database from certain users. Users are not aware of any hidden attributes (columns) or rows that are missing from the view.
- A view can be defined over one or more base table(s) with a user being granted the appropriate privileges to use the view, but not to use the base table(s). In this way, a view is more restrictive than simply granting privileges to a user on the base table(s). For example, it provides additional mechanisms to restrict access to specific rows and/or columns of the base table(s).
- A view permits users to access data in a customized way, so that the same data can be seen by different users in different ways, at same time.
- It can simplify complex operations on base relations. For example, queries involving joins are defined once in a view and the user will then be able to construct simpler queries using the view.
- Views also provide logical data independence. For example:
    - If a new attribute (column) is added to a relation (table), existing users can be unaware of its existence if their views are defined to exclude it (they do not need to see the new column). Logical data independence means there is no need to update the view because a new column has been added to the conceptual schema.
    - If an existing table is rearranged or split up into smaller tables, the view can be updated to include the necessary columns from the smaller tables using a join, so that the users can continue to see their original views. Logical data independence here means that although the conceptual schema has changed, the queries for the external views can be updated such that the user is unaware of any underlying changes.

Advantages and Disadvantages of Views[1]

Advantages

In the case of a DBMS running on a standalone PC, views are usually a convenience, defined to simplify database requests. However, in a multi-user DBMS, views play a central role in defining the structure of the database and enforcing security and integrity. The major advantages of views are described next.

Data independence

A view can present a consistent, unchanging picture of the structure of the database, even if the underlying source tables are changed (for example, if columns added or removed, relationships changed, tables split, restructured or renamed). If columns are added or removed from a table, and these columns are not required by the view, then the definition of the view need not change. If an existing table is rearranged or split up, a view may be defined so that users can continue to see the old table. In the case of splitting a table, the old table can be recreated by defining a view from the join of the new tables, provided that the split is done in such a way that the original table can be reconstructed. We can ensure that this is possible by placing the primary key in both of the new tables.

Currency

Changes to any base tables in the defining query are immediately reflected in the view.

Improved security

Each user can be given the privilege to access the database only through a small set of views that contain the data appropriate for that user, thus restricting and controlling each user's access to the database.

Reduced complexity

A view can simplify queries, by drawing data from several tables into a single table, thereby transforming multi-table queries into single-table queries.

Convenience

Views can provide greater convenience to users as users are presented with only the part of the database that they need to see. This also reduces the complexity from the user's point of view.

Customisation

Views provide a method to customise the appearance of the database, so that the same underlying base tables can be seen by different users in different ways.

Data integrity

If the WITH CHECK OPTION clause of the CREATE VIEW statement is used, then SQL ensures that no rows that fail to satisfy the WHERE clause of the defining query is ever added to any of the underlying base table(s) through the view, thereby ensuring the integrity of the view.

[1] Source: [© Pearson - Database Systems: A Practical Approach to Design, Implementation, and Management - http://app.kortext.com/read/80118/?page=258-259]

Disadvantages

Although views provide many significant benefits, there are also some disadvantages with SQL views.

Update restrictions

In some cases, a view cannot be updated. See notes on Updateable Views below.

Structure restrictions

The structure of a view is determined at the time of its creation. If the defining query was of the form SELECT * FROM …., then the * refers to all the columns from the base table present when the view is created. If columns are subsequently added to the base table, then these columns will not appear in the view unless the view is dropped and recreated.

One of the questions in today's lab tutorial exercise demonstrates this.

Performance

There is a performance penalty to be paid when using a view. In some cases, this will be negligible, while in other cases it may be more problematic. For example, a view defined by a complex, multi-table query may take a long time to process, as the view resolution must join tables together *every time the view is accessed*. View resolution requires additional computer resources. An alternative approach to maintaining views to overcome this is to use view materialisation (or simply temporary tables).

View resolution is the process of looking up the definition of the view and translating the query into an equivalent query request against the source tables before performing this equivalent query request. In view materialisation, the view is stored as a temporary table in the database when the view is first queried. Thereafter, queries run against the materialised view (temporary table) can be much faster than recompiling the view each time. The difficulty of this alternative approach, however, is in maintaining the currency of the data in the temporary table while the base table(s) are being updated.

Restrictions on Views

Although standard ISO SQL imposes a number of restrictions on the creation and use of views, there are considerable variations among DBMSs. Thus, what is allowed in one DBMS may not necessarily be allowed in another. For example, the ISO SQL standard imposes the following restrictions (all which MS SQL Server allows):

- If a column in a view definition is based on an aggregate function:
    - The column may only be used in the SELECT list and ORDER BY clause of queries using the view.
    - The column may not be used in the WHERE clause or be an argument of an aggregate function.

  The following queries will fail according to the ISO standard (but will work in MS SQL Server):

```
SELECT COUNT(numProperties) FROM StaffPropertyCount;
```

```
SELECT * FROM StaffPropertyCount WHERE numProperties > 2;
```

- Grouped views may never be joined.

Updateable Views

Not all views are updateable. There are restrictions on the types of data updates that can be made through views and generally:

- Updates are allowed if the columns being updated are from a single base table and contain either the primary key or a candidate key of the base table.
- Updates are not allowed through views involving multiple base relations.
- Updates are not allowed through views involving aggregation or grouping operations.

Thus, views are classed as theoretically not updateable, theoretically updatable or partially updatable.

ISO specifies that a view is updateable if and only if:

- DISTINCT is not specified in the defining query.
- Every element in the SELECT list of defining query is a column name and no column appears more than once (columns cannot constant values, derived or based on a computation).
- The FROM clause specifies only one table. If the source table is a view it must also satisfy these conditions.
- The WHERE clause does not include any nested SELECTS that reference the table in the from clause.
- There is no GROUP BY or HAVING clause in the defining query.
- Added data does not violate integrity constraints of base table.

Note: The primary key or candidate key is needed to update/delete specific rows. In the case of inserts, the primary key is also needed if the primary key is not an auto increment key.

See Furtado A. and Casanova M. (1985). Updating relational views. In Query Processing in Database Systems

View Resolution

Having considered how to create and use views, we now look more closely at how a query on a view is handled. To illustrate the process of view resolution, consider the following query, which counts the number of properties managed by each member of staff.

This query is based on the StaffPropertyCount view example.

SELECT StaffNo, NumProperties FROM StaffPropertyCount WHERE branchNo = ‘B003’ ORDER BY staffNo;

View resolution merges the example query with the defining query of the StaffPropertyCount view as follows:

1. The view column names in the SELECT list are translated into their corresponding column names in the defining query. This gives:
   SELECT s.BranchNo, s.staffNo AS staffNo, COUNT(*) AS NumProperties
2. View names in the FROM clause are replaced with the corresponding FROM lists of the defining query: FROM Staff s JOIN Property p ON s.StaffNo = p.ManagedBy
3. The GROUP BY and HAVING clauses are copied from the defining query. In this example, we have only a GROUP BY clause: GROUP BY s.branchNo, s.staffNo
4. The final merged query (view resolution) becomes:
   SELECT s.staffNo AS staffNo, COUNT(*) AS NumProperties
   FROM Staff s
   JOIN Property p ON s.StaffNo = p.ManagedBy
   GROUP BY s.branchNo, s.staffNo

View Materialisation[2]

In the previous section we discussed one approach to handling queries based on a view, in which the query is modified into a query on the underlying base tables. One disadvantage with this approach is the time taken to perform the view resolution, particularly if the view is accessed frequently. An alternative approach, called view materialization, is to store the view as a temporary table in the database when the view is first queried. Thereafter, queries based on the materialized view can be much faster than recomputing the view each time. The speed difference may be critical in applications where the query rate is high and the views are complex, so it is not practical to recompute the view for every query.

Materialized views are useful in new applications such as data warehousing, replication servers, data visualization, and mobile systems. Integrity constraint checking and query optimization can also benefit from materialized views. The difficulty with this approach is maintaining the currency of the view while the base table(s) are being updated. The process of updating a materialized view in response to changes to the underlying data is called view maintenance. The basic aim of view maintenance is to apply only those changes necessary to the view to keep it current.

Review Questions

- Discuss the advantages and disadvantage of views.
- How can views be used as a security mechanism in a Database?

[2] Source: Adapted from [© Pearson - Database Systems: A Practical Approach to Design, Implementation, and Management - http://app.kortext.com/read/80118/?page=260]