VIEW MANAGEMENT IN DISTRIBUTED DATA BASE SYSTEMS

E. BERTINO, L.M. HAAS, B.G. LINDSAY

IBM San Jose Research Lab
5600 Cottle Road
SAN JOSE, CA95193 USA

1. Introduction

The structure of data to be stored by a Data Base Management System (DBMS) is usually decided by a database administrator. Individual users and applications are generally interested in only a subset of the data stored in the database. Often, they wish to see this subset structured in a way which reflects their particular needs. Since it is not generally possible to structure a database so as to please all of its users, some mechanism is needed whereby each user can view the data according to his (her) own requirements. The representation of the data structure as seen by a user is often referred to as an external schema; the view mechanism is a means by which a DBMS can support various external schemas.

Besides providing users with tailored views of the data, the view mechanism contributes to [Chamberlin75]:

- Data independence: giving applications a logical view of data, thereby isolating them from data reorganization.

- Data isolation: giving the application exactly that subset of data it needs, thereby minimizing error propagation.

In a relational DBMS, a view is defined as a "virtual table" derived by a specific query on one or more base tables. The relational operations join, restrict and project as well as statistical summaries of tables may be used to define a view. Access rights may be granted and revoked on views just as though they were ordinary tables. This allows users to selectively share data, preventing unauthorized users from reading sensitive information.

SQL/DS [SQL81] and INGRES [Stonebraker76] are examples of relational DBMS's that provide the view facility.

In this abstract, the implementation of views in distributed relational DBMS's is discussed. Section 2 briefly reviews view management in a single-site DBMS. In section 3, the view concept is extended to a distributed DBMS (DDBMS) and two types of views are introduced: shorthand views and protection views. Finally section 4 outlines view management in a DDBMS.

2. Views in Single-Site Database Systems

A DBMS must construct and store an internal representation for each view that it supports. This occurs at view definition time. In systems such as SQL/DS, this internal representation is a parse tree for the view definition statement. When a view is later referenced in a query, a view composition operation is performed to combine the view's parse tree with the query parse tree. The result is a composite parse tree which only contains references to real stored tables.

In creating the internal representation of a view, names in the view definition statement are bound to the specific objects they reference, namely tables and other views. A view is therefore logically dependent on the continued existence of all objects that it references. If an object is dropped or substantially changed (for example, if the columns of a table are rearranged or a view is redefined) then the views referencing those objects must be invalidated.

In addition, since at view usage time the operations performed on the view will be translated into operations on the underlying objects, the view must have the necessary privileges on those objects. The privileges are granted to the view by the view definer. The view definer must therefore have the necessary privileges on the objects referenced in the view definition to be able to define the view. This allows users who do not have privileges on the underlying objects, but who do have privileges on the view, to access those objects through the view, according to protection requirements expressed by the view. If all privileges on an underlying object are revoked from the view definer, the view is no longer valid.

To be certain that a view is valid, the DBMS must keep track of dependencies of views upon all objects they reference. At view definition time, the system determines the set of objects referenced by the view, and the set of authorizations needed by the view. Records of the form <view name>

depends on <object name> and <view name> *possesses* <authorization> *on* <object name> are stored in special system catalogs. If an object is dropped or substantially changed, or if an authorization is revoked, the catalogs can be searched to find the views affected by the change, and the views can be marked invalid.

The next time the view is used in a query, the system notes the invalid state of the view, and drops the view. Alternatively, it could attempt to revalidate the view, by redefining the view against the current database. Note that revalidation will not always succeed: if an object referenced by the view has been dropped, the view cannot be defined. If, however, the object has merely been changed, and the changes are compatible with the view definition, the view can be revalidated (for example this might happen if the columns of a table have been reordered).

3. Views in Distributed Database Systems

Within the last few years, distributed database management systems (DDBMS) have become a rapidly growing field of investigation and a number of implementations have been reported [Williams81,Rothnie80,Stonebraker77]. Among the numerous goals of a DDBMS, two have been recognized as key objectives: site autonomy and data distribution transparency.

Site autonomy means that each site can operate on its own data as a stand-alone, single-site DBMS, and that each site retains local control of its own data, even if the site participates in the execution of a distributed query. This guarantees better resiliency to failures of sites and communication lines, since there are no centralized functions or services, such as a global dictionary or centralized deadlock detector. Further, each site performs all operations on its own local data, including authorization checking and, of course, database accesses and updates.

The second objective, distribution transparency, means that users are shielded from the physical distribution and redundancy of data and are able to interact with the distributed system as easily as with a conventional centralized one. This ensures logical independence of applications.

Any extension of views to a DDBMS must preserve the appearance of views as virtual tables. This presents many problems due to the fact that views in a DDBMS may be defined using tables which are not local to the view definition site and/or using other views defined at remote sites. Views defined using non-local objects are themselves distributed objects, which requires that the operations of creating, dropping and using a view be distributed operations.

The issue of authorization on views also has major implications for the implementation. When a view is used in a query, view composition must take place in order to derive an execution strategy for the query. If views are not objects of authorization, this composition can take place at any site. If views are objects of authorization, site autonomy considerations require that the view definition site maintain control over the materialization of the view. In particular, view composition must take place at the view definition site. (Actually any "trusted" site will do, but the "trusted" set should default to the view definition site). If view composition is allowed to occur at any site, a malicious site could pervert the view definition by dropping restrictions or projections in the view definition.

Forcing view composition at the view definition site can have negative effects on performance. If the definition site of a view is not the site at which a query using the view is submitted (the query master site), then it may not be possible for a single site to produce a complete execution strategy for the query, because the view definition may only be composed into the query at the view definition site. Further, in systems which separate planning a query from its execution, the execution strategy must include accessing the view definition site to check that the user is still authorized to use the view. This must be done at execution time even if no tables referenced in the fully composed query are stored at the view definition site.

For these reasons, the execution strategy for a query referencing a remotely defined view is unlikely to be optimal, and a performance penalty may be incurred. If views were not objects of authorization the query site could produce a complete execution strategy, which would not require accessing the view definition site unless some table were actually stored there.

Hence, two types of views are recognized. Shorthand views provide the data hiding, data conversion, typing elimination and renaming functions associated with views, but are not objects of authorization. The user posing queries against a shorthand view must be authorized to access the objects referenced by the view. Protection views provide the same semantics as shorthand views and in addition are objects of authorization. Authorization to access the objects referenced by the protection view belongs to the view and the user of the view only needs the privilege to use the protection view. Queries referencing remotely defined shorthand views will in general execute more efficiently than identical queries with shorthand views replaced by protection views.

4. Distributed View Management

As mentioned in the previous section, views are defined in terms of queries which may reference local and non-local tables and views. Queries may be imbedded in programs, which are precompiled by (D)DBMS's such as System R [Astrahan76] and R* [Wiliams81]. The result of precompilation is a set of access modules, defining the execution plan for the query, which are stored in the (distributed) database. Precompilation also creates dependencies (as described in section 2) for the program on the tables and views referenced in the program. Thus, if a table is dropped or a view redefined, the program must be invalidated. At execution time of a program, the system checks whether the program is valid. If the program is still valid, the system loads and executes the necessary access modules. If the program has been invalidated, the system may try to recompile the program; if recompilation succeeds, the program is executed, otherwise an error is reported [Ng82].

Since distribution transparency requires that the system have the same behavior with respect to users as a centralized DBMS, a correct implementation of views in DDBMS must ensure that views are dependent on the objects they reference, and that programs referencing views are dependent on

those views. These requirements ensure a consistent usage of views by users.

In addition, site autonomy requires that each site be able to perform any action on local (non-distributed) objects, such as dropping local tables or purely local views, without notifying any other site. In particular, it should not be necessary to contact other sites at which programs or views referencing the local objects are stored or defined.

These two requirements suggest that dependency recording should be distributed among the sites of those objects on which the view (or program) depends. In other words, view or program dependencies on remote tables should be recorded at the sites where those tables are stored. This allows local invalidation of distributed programs or remotely defined views if a local table is dropped or changed.

Actually the situation is somewhat more complicated. Programs and views may depend on remotely defined protection and shorthand views, as well as remotely stored tables. Dependencies on protection views are recorded at the view definition site. In fact this site will be accessed at execution time, since the view is materialized at that site.

If the dependency of a program on a shorthand view is recorded at the view definition site, and that view is later dropped, it may not be possible to invalidate the program. This will happen if no table referenced by either program or view is stored at the view definition site. In this situation, the program has no need to access the view definition site at execution time and hence it will not discover that the view has been dropped.

To invalidate a program in these circumstances, dependencies on shorthand views are recorded at other sites, chosen in such a way that any program referencing the view must access these sites at execution time. In fact, those sites are the sites at which tables referenced by the view are stored.

An obvious consequence of these distributed dependencies is that view definition and view drop are distributed operations. At view definition time, the dependency of a view on remote tables must be recorded at the table store sites. When dropping a view, remote sites storing view dependencies must be accessed in order to delete these dependencies and, at the same time, invalidate programs and views depending upon the dropped view. Note that remote sites are only accessed when the view is a distributed object. Local views are still dropped locally.

5. Conclusions

In this paper, the implementation of views in a DDBMS has been discussed. Two kinds of views have been introduced: shorthand views and protection views.

The approach chosen for view implementation has many advantages. Programs using views depend upon them; thus they always correctly incorporate the current semantics of any view referenced by the programs. If the views are dropped or changed, the programs will be invalidated. Views also depend on the objects referenced in the view definition statements; this allows invalidation of views when one of these objects is dropped or changed.

View definition and drop view are distributed operations, involving all sites that store objects referenced in the view definition statement. Since the actions of defining and dropping views are expected to occur relatively infrequently, this is not a high price to pay for good query performance with these semantics.

REFERENCES

[Chamberlin75] Chamberlin,D.D., Gray,J.N. and Traiger,I.L. , "Views, Authorization and Locking in a Database System", Proc. AFIPS NCC, Vol. 44, 1975.

[Ng82] Ng, P., "Distributed Compilation and Recompilation of Database Queries", IBM Research Laboratory RJ3375 San Jose, Calif., January 1982.

[Rothnie80] Rothnie,J.B, Bernstein,P.A., Fox,S.A, Goodman,N., Hammer,M.M., Landers, T.A., Reeve,C.L., Shipman,D.W. and Wong,E., "A system for distributed database (SDD-1)", ACM Transactions on Database System, March 1980.

[SQL81] IBM Corp., "SQL/Data System: Application Programming", SH24-5018, 1981.

[Stonebraker76] Stonebraker,M., Wong,E.,Kreps,P., and Held, G., "The Design and Implementation of INGRES", ACM Trans. Database Syst. 1,3 (Sept.1976).

[Stonebraker77] Stonebraker,M., Neuhold,E., "A Distributed Version of INGRES", Proc. 2nd Berkeley Workshop Distributed Data Management and Computer Networks, May 1977.

[Wiliams81] Wiliams,R. et Al, "R*: An Overview of the Architecture" Proceedings of the international Conference on Database Systems, Jerusalem, Israel, June 1982. Published in Improving Database Usability and Responsiveness, P.Scheuermann, ed. Academic Press, N.Y.