Module 4: Semantic Data Control

Introduction to Semantic Data Control

All databases, whether they are centralized and distributed, must perform semantic data control. This means that users will perform correct operations on the database for which they are authorized. Semantic Data Control ensures that users can only read, insert, update, and delete data for which they are authorized and that the operations are performed correctly. Three methods of semantic data control will be covered in this module: view management, security constraints, and semantic integrity constraints. Violation of one of these will result in an operation being rejected.

We will first review centralized database solutions. Then, we will review some complications imposed by distributed database systems. For example, the database catalog must be used to store the rules. A distributed database catalog induces more complexity. The complexity revolves around whether to duplicate the representation of the rules or to distribute them. We will see more on this shortly.

View Management

A view is a virtual relation that is not materialized like a base relation is materialized (stored). A view draws its information from materialized relations. In a sense, views are dynamic windows onto the materialized relations. External schemas that we discussed in Module 2 can be implemented as views.

Views can also be used to implement data security. Views present only the information to users that they are authorized to see and hide the information that they are not authorized to see.

Assume an employee table EMP that has columns including the employee number (eno), the employee name (ename), a title, and a salary. Suppose that we wish to create a view called SYSAN that lets us view only the employee numbers and names of system analysts. One definition of SYSAN is:

```
CREATE    VIEW      SYSAN(eno, ename)

AS        SELECT    eno, ename
          FROM      EMP
          WHERE     title = 'System Analyst'
```

The result of querying the SYSAN view may be:

ENO	ENAME
E2	M. Smith
E5	B. Casey
E8	J. Jones

If a user is authorized to view information only about system analysts from the EMP table, the user is granted privileges on the SYSAN view and not the EMP table. Nevertheless, the SYSAN view can be treated like any other table and can be used in another query as if it were a table.

```
SELECT    SYSAN.ename, ASG.pno, ASG.resp
FROM      SYSAN, ASG
WHERE     SYSAN.eno = ASG.eno
```

A DBMS that runs this query must map the query expressed using the SYSAN view to a query expressed on base relations. This process is called *query modification*. The DBMS would change the query that joins SYSAN to ASG to this query that joins EMP to ASG:

```
SELECT    EMP.ename, ASG.pno, ASG.resp
FROM      EMP, ASG
WHERE     EMP.eno = ASG.eno AND
          EMP.title = 'System Analyst'
```

Example results may be:

ENAME	PNO	RESP
M. Smith	P1	Analyst
M. Smith	P2	Analyst
B. Casey	P3	Manager
J. Jones	P4	Manager

Dynamic Views

Views can be dynamic based on some system or environment variable. For example, suppose the $USER variable takes on the value of the user who is logged in. The following query will yield different results for different users:

```
CREATE    VIEW      ESNAME
AS        SELECT    E2.eno, E2.ename, E2.title
          FROM      EMP E1, EMP E2
          WHERE     E1.title = E2.title  AND
                    E1.eno = $USER
```

The query employs tuple variables to find employees who have the same title as the current user. Tuple variables E1 and E2 in the query above are employed to provide a different conceptual instance of the *EMP* table. It is as if the query joins two identical copies of the *EMP* table. By running the following query, users will return the names of employees who have the same title as the current user.

```
SELECT    *
FROM      ESNAME
```

View Updates

Updating the database through views cannot be done in general. Updates through views will only work when the view enables the user to provide all primary key, foreign key, and non-null column values.

As an example, a user can insert data into the database via the SYSAN view, which does not contain the TITLE column, provided that the TITLE column value can have a NULL value. If the DBMS was really clever, it could provide the string 'System Analyst' as the value of the TITLE column. However, DBMS systems cannot provide the values of unspecified columns in general.

In the following example, a user cannot insert a new row into this view if the ENO column is the primary key of the EMP table. This is because primary key columns must contain some value if it is to be inserted into the EMP table. However, VIEW1 does not provide an opportunity to specify ENO.

```
CREATE    VIEW      VIEW1
AS        SELECT    ename, title
          FROM      EMP
```

Similarly, a user cannot insert a new row into the following view because it does not provide a user with the ability to specify ENO, which is the primary key of the EMP table and the foreign key of the ASG. While the view is defined using a join on ENO, it only provides users with a view of the ENAME, PNO, and RESP columns.

```
CREATE    VIEW      VIEW2
AS        SELECT    EMP.ename, ASG.pno, ASG.resp
          FROM      EMP, ASG
          WHERE     EMP.eno = ASG.eno AND
                    EMP.title = 'System Analyst'
```

View Management in Distributed Databases

In distributed database systems, views are defined on the database tables and fragments. As in single database systems, views may be used as base relationships so they are stored in the database catalog similarly. However, there are a number of issues that must be decided.

One issue is that the distributed database systems must decide which of the distributed sites to store view definitions. One option is to store them on the same sites where the referenced tables and fragments reside. This is a good solution if the views can be co-located with all their corresponding tables and fragments. Issues arise when the views refer to tables and fragments that are not co-located. Suppose a view is defined on fragments A and B. Fragment A is stored at one site while

fragment B is stored at another site. The view can be stored with fragments A and point to fragment B. When the view is resolved, a cross-site join is required to solve the query on the view. Some suggest that views should be tailored to ensure that they correspond to fragments on one site and their definitions should be co-located with those fragments. This means that one could not define views that join distributed fragments. Instead, queries that join distributed fragments would join the results of distributed views.

Alternatively, fragments can be replicated on multiple sites so that the views can be replicated where fragments are replicated. There are still issues with this arrangement because distributed views are expensive to maintain. Each time data in a fragment is updated, each of the distributed fragments need to be updated as well.

Accessing views on distributed fragments is costly. An important advantage of views in general is that their query execution plans are generally precompiled for better performance. Pre-compilation improves the performance of distributed views as well.

Another solution is to calculate the views on a periodic basis, produce a snapshot of the data, and write the snapshot into a single table. When a calculated view is actually referenced, the results from the snapshot are returned rather than from an executed join. An important trade off is determining the right frequency to calculate the snapshots. Frequent snapshots are likely to be more accurate, but more system resources are used. Less frequent snapshots are less likely to be accurate, but less system resources are used. The right frequency depends on the requirements of the applications using the database.

Data Security

Data Security is *data protection* and *authorization control*. Data protection denies unauthorized access to data. Authorization control denies users access to database resources for which they are not authorized. In a distributed DBMS, multiple sites must protect data from multiple users.

Single Database System Approach

Single database systems control access to resources by storing tuples that declare what operations users have the right to run against some database object. The format of a tuple is <user, operation, object>. This meaning of a tuple is that a *user* has the right to perform a specific *operation* on an *object*. A DBMS validates a user (e.g., at login, during a remote call, etc.) via a user-password pair <user, password>.

What is an object? Objects can be broadly defined to mean any information set that can be extracted from the database. This can be a simple element such as the value of a column of a row (e.g., John's name), a group of column values (e.g., a list of the ages of people that are Systems Analysts), or a complex derivation from the database (e.g., the sum of the salaries of people that have worked at a company for 10 years). Following are more sample objects:

- A relation, a set of columns, or a set of rows constrained by some criteria (i.e., WHERE clause predicates)

- An arbitrarily complex object (e.g., a list of employee names where eno = salary/1000)

Grant and Revoke Policies

The SQL syntax for granting and revoking privileges is:

- GRANT <privilege> ON <object> TO <user>
- REVOKE <privilege> FROM <object> TO <user>

where:

- <privilege> = insert, update, delete, select, grant
- <user> = public, any user name

The privileges are stored as a table of triples in the form <user, privilege, object>. In addition, a list of list of grantors is stored to keep track of who gave whom privileges. Initially, the DBA gives GRANT privileges to other users. Those users, in turn, can grant privileges to other users based on the privileges the original users have. If a user does not have the authority to grant a privilege, then he/she cannot grant that privilege to other users. A user can revoke the privileges of other users that the original user granted. A user can revoke a privilege that he/she granted to a second user. If the second user granted that same privilege to a third user, then when the first user revokes the second user's privilege, the system automatically cascades the revocation by revoking the privilege from the third users. A user cannot revoke a privilege that he/she did not grant directly or indirectly.

(Watch Module 4A video – Cascading Revokes)

Distributed Database Issues

Remote user authentication and distribution of authorization rules are the most significant data security issues of distributed database systems.

Since users interact with data at distributed locations, there is the need for remote authentication. Each site that with which a user interacts must perform authentication of that user. One approach is for each site to maintain a directory of all users and passwords enabling each site to perform authentication locally. A drawback is that every site must be kept current of users and passwords, which does incur some communications costs and synchronization costs.

The other approach is for each site to support authentication for its own local users. When a site receives an authentication request from a user whose authentication information is maintained at another site, the current site must generate a remote request to authenticate the user. There is some cost to maintaining a directory of which site is responsible for maintaining the authentication of which users. There is also a cost for each remote request. A benefit of this approach is that the maintenance of user authentication is simplified because each user is managed by one site only.

Authorization rules that maintain GRANT and REVOKE privileges can be replicated throughout the distributed database system or they can be maintained at one site. Fully replicated rules can be processed rapidly at the time of query access or modification (insert, update, delete). Furthermore, since the rules are replicated, each site can maintain a compiled version of the privileges, which can enable rapid verification.

The other approach is to maintain distributed authorization rules. These are costly to maintain in general. However, if the locality of reference of the rules are high, which means that the rules are kept near the fragments that they operate on, then better performance can be achieved.

Views are treated like relations. If the authorization is replicated, view management is easy. If the authorization is distributed, view management becomes more costly if the rules are defined on distributed fragments.

Semantic Integrity Control

How does one guarantee database consistency? A database is consistent if it satisfies *semantic integrity constraints*, which is a collection of knowledge about the intended use of the database to ensure consistency. The enforcement of these constraints ensures that a database does not end up in inconsistent state.

The two types of constraints are *structural* and *behavioral* constraints. Structural constraints are based on database models. They ensure that columns whose values are to be unique remain unique. They ensure that one-to-many relationships between primary and foreign keys are maintained.

Behavioral constraints describe what types of data should be stored in the data columns and how they need to remain consistent with the rest of the database. They are how the database should behave when users operate on the data. In some cases, behavioral constraints can be difficult to process.

In a central database, integrity constraints are represented as assertions in tuple relational calculus. They are of the form:

- For all (∀) tuples of some type, some constraint must be true (universal quantification)
- There exists (∃) a tuple of some type that has a constraint on its value (existential quantification)

Here is a list of some constraints:

Structural constraints:

- Non-null: ENO not null in EMP
  - Make sure that every value of ENO in EMP is not null
- Unique key:
  - (ENO, PNO) unique in ASG
  - Make sure that the combination of ENO and PNO are unique in ASG
- Foreign key:

    - PNO in ASG references PNO in PROJ
    - Make sure that every PNO value in ASG has a corresponding PNO value in PROJ
- Functional dependency:
    - ENO in EMP determines ENAME in EMP
    - The value of ENAME in EMP is dependent ENO in EMP

Behavioral constraints:

These are defined using the following syntax:

CHECK ON <relation> WHEN <update type> (qualification)

- Domain Constraint:
    - CHECK ON PROJ (BUDGET >= 500K & BUDGET <= 1M)
    - Make sure that all BUDGET in PROJ are between $500K and $1M

- Domain Constraint on Deletion:

    - CHECK ON PROJ WHEN DELETE (BUDGET = 0)
    - A row of PROJ cannot be deleted unless its BUDGET is $0

- Transition control:

    - CHECK ON PROJ (NEW.BUDGET>OLD.BUDGET AND NEW.PNO = OLD.PNO)
    - A new BUDGET has to be larger than an old BUDGET and a new PNO has to be larger than an old PN

- Functional dependency

    - CHECK ON e1:EMP, e2:EMP (e1.ENAME = e2.ENAME IF e1.ENO = e2.ENO)
    - Given two views of the EMP table, make sure that if a two rows share the same ENO then they share the same ENAME as well

- Constraint with aggregate function

    - CHECK ON g:ASG, j:PROJ (SUM(g.DUR WHERE g.PNO = j.PNO) < 100 if j.PNAME = 'CAD/CAM')
    - For the 'CAD/CAM' project, make sure that the sum of the durations of people assigned to that project is less than 100

Enforcing Semantic Integrity Control

(Watch Module 4B video – Enforcing Semantic Integrity)

Distributed Semantic Integrity Control

As in centralized databases, distributed integrity assertions are specified in tuple relational calculus. However, there are different strategies for enforcing the assertions based on the type of the assertion. The three categories of assertions are:

- Individual assertions: single-relation & single-variable
- Set-oriented assertions: single-relation and multivariable constraints
- Aggregate assertions: special processing

Remember that relations are fragmented so the assertions are also fragmented. With this in mind, the following video discusses these strategies.

(Watch Module 4C video – Enforcing Semantic Distributed Integrity)