NCSC TECHNICAL REPORT - 005
Volume 5/5
Library No. S-243,039

FOREWARD

This report is the fifth of five companion documents to the *Trusted Database Management System interpretation of the Trusted Computer System Evaluation Criteria.* The companion documents address topics that are important to the design and development of secure database management systems, and are written for database vendors, system designers, evaluators, and researchers. This report addresses discretionary access control issues in high assurance secure database management systems.

[illegible]

Keith F. Brewster
Acting Chief, Partnerships and Processes

May 1996

ACKNOWLEDGMENTS

The National Computer Security Center extends special recognition to the authors of this document. The initial version was written by Vinti Doshi, Sushil Jajodia, and LouAnna Notargiacomo of the MITRE Corporation. The final version was written by Gary Smith, Doug Landoll, Bill Wilson, and David Wichers of Arca Systems, Inc.

The documents in this series were produced under the guidance of Shawn P. O'Brien of the National Security Agency, LouAnna Notargiacomo and Barbara Blaustein of the MITRE Corporation, and David Wichers of Arca Systems, Inc.

We wish to thank the members of the information security community who enthusiastically gave of their time and technical expertise in reviewing these documents and in providing valuable comments and suggestions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

This document is the fifth volume in the series of companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria* [TDI 91; DoD 85]. This document examines discretionary access control (DAC) issues in high assurance secure database management systems and summarizes the research to date in this area.

1.1 BACKGROUND AND PURPOSE

In 1991 the National Computer Security Center published the *Trusted Database Management System Interpretation* (TDI) of the *Trusted Computer System Evaluation Criteria* (TCSEC). The TDI, however, does not address many topics that are important to the design and development of secure database management systems (DBMSs). These topics (such as inference, aggregation, and database integrity) are being addressed by ongoing research and development. Since specific techniques in these topic areas had not yet gained broad acceptance, the topics were considered inappropriate for inclusion in the TDI.

The TDI is being supplemented by a series of companion documents to address these issues specific to secure DBMSs. Each companion document focuses on one topic by describing the problem, discussing the issues, and summarizing the research that has been done to date. The intent of the series is to make it clear to DBMS vendors, system designers, evaluators, and researchers what the issues are, the current approaches, their pros and cons, how they relate to a TCSEC/TDI evaluation, and what specific areas require additional research. Although some guidance may be presented, nothing contained within these documents should be interpreted as criteria.

These documents assume the reader understands basic DBMS concepts and relational database terminology. A security background sufficient to use the TDI and TCSEC is also assumed; however, fundamentals are discussed whenever a common understanding is important to the discussion.

1.2 SCOPE

This document addresses DAC issues in high assurance secure DBMSs. It is the fifth of five volumes in the series of TDI companion documents, which includes the following documents:

- *Inference and Aggregation Issues in Secure Database Management Systems* [Inference 96]
- *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems* [Entity 96]
- *Polyinstantiation Issues in Multilevel Secure Database Management Systems* [Poly 96]
- *Auditing Issues in Secure Database Management Systems* [Audit 96]
- *Discretionary Access Control Issues in High Assurance Secure Database Management Systems*

This series of documents uses terminology from the relational model to provide a common basis for understanding the concepts presented. For most of the topics covered in this series the concepts presented should apply to most database modeling paradigms, depending on the specifics of each model.

1.3 INTRODUCTION TO HIGH ASSURANCE DAC

DBMSs are large, complex software packages which provide an enterprise with significant capabilities to manage data. Most commercial DBMSs are based on the relational model [Codd 70]. The Structured Query Language (SQL) is the standard interface language for relational DBMSs and contains the functionality to specify application-dependent access controls (in particular, DAC) to database objects using the GRANT or REVOKE statements. SQL also provides the capability to control access to data through *views.* A view is a derived relation, normally a subset of a database, that is specified with a view definition written in SQL or another data description language.

High Assurance systems are defined as those that meet the TCSEC requirements for class B2 and above. The TCSEC specifies both functionality and assurance requirements for each evaluation class. Although additional DAC functionality is added at B3, the difficult problem standing in the way of DBMSs intended to meet higher level TCSEC requirements is the need to meet the TCSEC assurance requirements. In addition to failing to meet the B3 TCB minimality and simplicity requirements, if the entire SQL engine is included within the TCB boundary, it is improbable that such an architecture would meet the additional B3 system architecture requirements of layering, abstraction, and data hiding. Such structuring would likely add considerable execution overhead to the product.

While there is not an explicit minimality requirement at B2, it would still be very difficult to incorporate a mechanism as large and complex as a SQL engine in the TCB while meeting B2 requirements. Most commercial DBMS architectures do not exhibit the internal structure and discipline necessary to meet the interpreted B2 modularity requirement. Furthermore, the B2 System Architecture requirement calls for “effective use of available hardware to separate those elements that are protection-critical from those that are not...” and requires that “...TCB modules shall be designed such that the principle of least privilege is enforced....” Also, the Design Documentation requirements include a description of how the “TCB implements the reference monitor concept... and is structured to enforce least privilege.” While recognizing that there are important distinctions between the B2 and B3 requirements, size and complexity of TCB mechanism is a major issue for both. (These TCSEC requirements are discussed in more detail in Section 2.1.)

Based on this background information, the DBMS high assurance DAC problem can be succinctly stated as:

- In most current DBMS software architectures the entire SQL engine is inside the TCB boundary since it contains the DAC enforcement mechanism.

- The SQL engine is typically too large and complex to meet the B2 and B3 assurance requirements.

Although the DBMS high assurance problem can be simply stated, there are many complex issues involved with providing acceptable solutions (both technically and from the user’s perspective).

These issues and research efforts to design high assurance DBMS DAC mechanisms are discussed in depth in the remaining sections.

1.4 AUDIENCES OF THIS DOCUMENT

This document is targeted at four primary audiences: the security research community, database application developers/system integrators, trusted product vendors, and product evaluators. In general, this document is intended to present a basis for understanding and discussion of the issues and techniques in obtaining high assurance DAC in DBMSs. Each of the specific audiences should expect to get the following from this document:

<u>Researcher</u>

This document describes the basic issues associated with high assurance DAC in DBMSs. Important research contributions are discussed as various topics are covered. By presenting current theory and debate, this discussion will help the research community understand the scope of the issue and highlight current alternatives.

<u>Database Application Developer/System Integrator</u>

This document highlights the difficulties of defining DAC policies within a DBMS. It describes the different issues of implementing DAC through (potentially overlapping) views. It also discusses the tradeoff between richness of policy and the degree of assurance which can be obtained for a mechanism enforcing the policy.

<u>Trusted Product Vendor</u>

This document describes the conflict between the degree of richness in DBMS DAC mechanisms and the attainable level of assurance. It then discusses approaches to implement high assurance DBMS DAC mechanisms and the benefits and drawbacks of these approaches. This document will provide a framework for understanding specific requirement interpretations as they are developed by the National Computer Security Center.

<u>Evaluator</u>

This document presents an understanding of high assurance DAC issues to assist the evaluation of DBMS enforcement of high assurance DAC.

1.5 ORGANIZATION OF THE DOCUMENT

The organization of the remainder of this document is as follows:

- Section 2 provides background by summarizing the TCSEC requirements for DAC in high assurance DBMSs, defining terminology, and describing DAC policies and mechanisms.

- Section 3 describes current practice for enforcing DAC in DBMSs with particular emphasis on DAC controls incorporated in SQL.

- Section 4 discusses critical issues associated with high assurance DAC including effectiveness of DAC, implications of using views for DAC, and the concept of balanced assurance.
- Section 5 presents approaches to providing high assurance DAC. This section addresses use of ACLs on database objects, gives an approach to implementing GRANT/REVOKE with high assurance, describes SeaView as an example of a balanced assurance approach, and discusses three approaches for providing at least limited view-based DAC with high assurance.
- Section 6 contains a summary.

SECTION 2

BACKGROUND

This section provides background requirements and terminology that forms the basis for the discussion of current database practice in Section 3, issues in Section 4, and the description of research efforts in Section 5. Section 2.1 presents the TCSEC feature and assurance requirements for DAC. Section 2.2 briefly describes typical OS DAC mechanisms. Basic relational database terminology is presented in Section 2.3.

2.1 TCSEC REQUIREMENTS

The TCSEC describes requirements for a secure operating system, with DAC as one of these requirements. The TDI extends these requirements to DBMSs. A formal definition of DAC as given in the TCSEC is as follows:

> "A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless constrained by mandatory access control)."

DAC mechanisms control a user's access based on the USER-ID, the access type, and the specific object being accessed.[1] The access type refers to the operation allowed to be performed on the particular object. Examples of different access types against database objects include SELECT, APPEND, and DELETE.

Security mechanisms can be implemented with varying degrees of assurance. The assurance of a mechanism is proportional to the effort needed to subvert the mechanism. Low-assurance mechanisms, are not analyzed as thoroughly nor implemented as carefully as high-assurance mechanisms and may be easier to subvert. The DAC requirements can be broadly categorized into two assurance areas, low and high assurance, which correspond to the DAC and associated assurance requirements stated in the TCSEC for class CI through BI (discussed in Section 2.1.1) and classes B2 through Al (discussed in Section 2.1.2), respectively. The TDI states that the DAC requirements for a DBMS apply as stated in the TCSEC to every TCB whose policy includes access control of subjects to named objects.

2.1.1 Requirements for Class C1 through B1

The DAC requirements for classes CI, C2, and BI define the minimum basic DAC requirements. The TCSEC states that for these classes, access control between named users and named objects should be defined and managed by the TCB. The TCSEC also states that the enforcement mechanism shall allow users to specify and control sharing of the named objects by named individuals or defined groups or both. Although the TCSEC does not formally define a named object, in this document a *named object* is referred to as any identifiable object that can be shared among users and the named

1. In a system enforcing both DAC and mandatory access control (MAC), the user must only be allowed access to an object when that access is permitted by both the MAC and DAC policies.

object need not uniquely identify information accessed through it. This definition of a named object is derived from the 1988 draft of the TDI [TDI 88].

For class C2 and above, the access controls must be capable of including or excluding access to the granularity of a single user. There is also a requirement for controls to limit the propagation of access rights. The question of how to limit propagation of access rights and how to provide a mechanism to revoke access rights is discussed further in Sections 2.3 and 3.1.2.

2.1.2 Requirements for Class B2 through A1

The DAC requirements for classes B2, B3, and A1 build upon those of the lower TCSEC classes with the following additions at B3:

- The discretionary access controls “shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object.
- ... “for each named object, it shall be possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given.”
- The TCSEC gives access control lists (ACLs) as the only example of a mechanism to satisfy these requirements at B3 as opposed to lower levels where self/group/public controls are included as an additional example mechanism.

Note that the TCSEC B3 requirements mention ACLs as an example (“e.g.,”). The specific mechanism for satisfying the requirement for DAC granularity to lists of single individuals or groups is implementation dependent. The additional DAC requirements imposed at the B3 level are relatively easy to satisfy given an SQL or equivalent implementation of the DAC policy. The difficulty in developing (and evaluating) high assurance DAC systems comes not from the B3 DAC requirements, but from the assurance requirements, namely system architecture, and design specification and verification requirements at the B2, B3, and A1 levels.

The additional assurance requirements affecting the implementation of the DAC mechanism at B2 are listed below:

- The TCB is based on a clearly defined and documented formal security policy model that requires DAC to be extended to all subjects and objects in the system.
- The TCB is internally structured into well-defined largely independent modules.
- The TCB makes effective use of available hardware to separate those elements that are protection-critical from those that are not.
- The TCB modules are designed such that the principle of least privilege is enforced.

At the B3 level, the requirements are modified to add:

- The TCB is designed and structured to use a complete, conceptually simple protection mechanism with precisely defined semantics.

- The protection mechanism plays a central role in enforcing the internal structuring of the TCB and the system.
- The TCB incorporates significant use of layering, abstraction, and data hiding.
- The TCB complexity is minimized, excluding from the TCB those modules that are not protection-critical.

Class Al additionally requires formal verification and documentation of the security policy model and top level design to assure that the discretionary access controls employed in the system are always invoked as part of the mediation for granting access.

2.2 OPERATING SYSTEM DAC MECHANISMS

Operating System (OS) DAC policies define protection on OS objects (e.g., files, directories). These policies may be implemented within an operating system in several ways. The operating system DAC mechanisms below are presented to allow comparison between them and DAC mechanisms employed in DBMSs. Note that it is possible for a DBMS to rely on these traditional OS DAC mechanisms to some degree (e.g., on the entire database, view definitions, tables) but not completely. Given the expected large set of users of a generic database, the DBMS itself must provide protection for fine granularity objects such as specific fields or values within a field in order to provide any reasonable level of performance.

2.2.1 Owner/Group/Other

A very common OS DAC mechanism is that of protection bits. Each operating system object has attached to it a set of bits for specifying access modes for different classes of users. The most common implementation includes the classes of owner, group, and all other users. The first set of bits specifies the read, write, and execute permissions for the owner of the file. The second set of bits specifies the read, write, and execute permissions for all users in the object's group. The last set of bits specifies the read, write, and execute permissions for all other users on the system. Because the protection bits are associated with the object, it is straightforward to determine which users have discretionary permission to access an object and to revoke those access rights when desired. Furthermore, propagation of access rights is directly controlled by controlling which user(s) can modify the protection bits.

The permission bits mechanism meets the requirement for allowing users to specify and control sharing of objects by named individuals or defined groups of individuals through the use of the group permission bits. For controlled sharing by named individuals, the group consists of just the named individual with which the object owner wishes to share.

At B3, the DAC requirements are strengthened to include controlled sharing of objects by named individuals **and** defined groups of individuals through the system capability to specify the respective modes of access or no access to named objects. The permission bits mechanism is limited to specifying a single set of access permissions to an object by a single group. As stated before, this group can specify a single individual and/or a defined group of individuals. However, this mechanism can give only a single set of access permissions to that group. Therefore, this mechanism is unable to specify the controlled sharing to individuals **and** defined groups of individuals at the same time and is unable to meet the B3 DAC requirements.

2.2.2 Capabilities

The capability mechanism associates an object access list (capability list) with each operating system subject.[2] This access list specifies each object the subject has access to and the set of access modes to these objects allowed for the associated subject. Because capability lists are associated with subjects, it may be difficult to determine which subjects possess access rights to a particular object at any given time. This can complicate revocation of access rights. Typically, a user may grant access to other users by providing a copy of the required capability. As a result, propagation of access rights is more complicated to control. The capability mechanism provides an efficient means of enforcing access control at runtime (e.g., It would be useful in DBMSs to be able to search the user/subject profile to see if the subject has a right to a given view of a specific table). However, this approach does require numerous entries per user for typical OSs and has difficulty with establishing who has access to individual files. This approach appears to be capable of satisfying the B3 DAC requirements.[3]

2.2.3 Access Control Lists

Access Control Lists (ACLs) are similar to capabilities, except that a subject access list is associated with each object. The access control list contains entries specifying the access permissions (and exclusions) for individual users or groups to the associated object. This mechanism is used in many operating systems and currently in all operating systems evaluated at B3 and higher. Because ACLs are associated with objects, revocation of access rights and control of propagation of access rights are easier than for capabilities. Of the three mechanisms described in this section, ACLs are considered to be the most flexible and are the predominant DAC mechanism used in current high assurance OSs.

2.3 RELATIONAL TERMINOLOGY

A database system is a collection of interrelated data, the *database,* and a set of programs to access that data, the *database management system (DBMS).* The database contains information about one particular enterprise. The primary goal of a DBMS is to provide an environment that is both convenient and efficient to use in retrieving information from and storing information into the database.

In a *relational* DBMS the data and the relationship among the data are represented by a collection of tables, each called a *relation.* A column in a relation is called an *attribute.* A row of a relation represents a relationship among a set of values and is called a *tuple.* Each data value in a row under a column is known as an *element.* Each relation in a relational database can be conveniently described by its *relation schema.* The schema corresponding to a relation defines the name of the relation and the names and data type information of all attributes that are contained in the relation.

2. Some operating systems have allowed applications to apply capabilities as a means of enforcing fine grained access control with application-specific mode interpretation to the objects they implemented and made available.
3. In order to specify a list of individuals and groups with their respective modes of access to an object, the operating system must search the capability lists of each individual or group for entries containing access to that object.

A *view*, which serves an important role in relational databases, is a *virtual* relation that is derived from other relations. The other relations can be either actual stored relations (sometimes called *base* relations to distinguish them from other view relations) or other view relations.

Views are desirable for two reasons. Sometimes it is not necessary for the users to have access to the actual stored relations. An organization’s security policy may require that certain data be hidden from a user or that only aggregate data be given to some users. Apart from security, certain applications may require the creation of smaller, more specific sets of relations, better matched to a specific users’ needs. Views can easily meet both requirements.

A view is specified by a *view definition* that is written in SQL or some other data description language. SQL is a standard query language for relational DBMSs which is also used to query and update the data managed by the DBMS [Date 89].

SECTION 3

CURRENT PRACTICE FOR DBMS DAC

This section describes current practice for providing DAC in DBMSs. It begins by describing the discretionary access controls defined within the SQL Standard [ANSI 92]. It then describes the internal structure of the typical SQL engine and concludes by describing how the SQL engine typically provides the DAC capabilities defined in SQL. This information provides a basis from which to compare the approaches for achieving high assurance DAC as described in Section 5.

3.1 SQL AND DAC

The SQL standard, which is used by most commercial DBMSs, includes specific requirements for enforcing DAC. There are three basic aspects to consider in SQL DAC:

1. What privilege types and access granularity may be specified,
2. How the DAC privileges are assigned, and
3. The use of views to further restrict access to underlying views or base tables.

These three aspects of SQL DAC are described in the following sections.

3.1.1 SQL DAC Privileges

In DBMSs, SQL is used to enforce DAC over database users' (or groups of users) access to database objects. These objects can be object definitions (e.g., database, table, or view definitions) or object contents (e.g., table or view contents). User access to these objects is defined in terms of privileges.

For object definitions, the following privileges are defined:

Create:	User is allowed to create additional objects of the appropriate type (e.g., create database, create table within specific database)
Drop:	User may delete specified object
Alter:	User may modify the definition of the specified object

For object contents, the following privileges are defined:

Insert:	User may create additional rows in the specified object
Delete:	User may delete rows from the specified object
Select:	User may retrieve values from the specified object
Update:	User may modify the data within the specified object
References:	User may specify a foreign key reference from the specified object schema to the foreign key of another object
Index:	User may create an index on the specified object

For stored procedures, or stored program units the following privilege is defined:

Execute:	User may invoke the specified object

These privileges are used to define a user's allowed mode of access to the associated object as a whole. Whenever a user performs a query (e.g., SELECT, INSERT, UPDATE, DELETE), the DBMS ensures that the user has the appropriate permissions to access all the objects referenced by the query.

3.1.2 Assigning SQL DAC Privileges

Given the privileges defined above, there must be a mechanism for assigning these privileges to users or groups of users and then revoking them when it is determined that the users or groups should no longer have them. This is accomplished through the use of the SQL GRANT and REVOKE statements. A particular concern is propagation of access rights, *i.e.,* whether a user given some privilege can then pass on those rights to other users. This is controlled through an option in the GRANT statement.

The user who creates a table in the database automatically becomes the owner of the table, and is therefore authorized to perform any operation on the table. By default, no other user can refer to the table unless the owner grants specific privileges to that user. The GRANT statement can be used to grant a privilege on a DBMS object to one or more users, groups, or roles. When the clause WITH GRANT OPTION is present in the statement, the users receiving the stated privileges have the additional authority to extend these privileges in turn to other users. The general SQL syntax for the GRANT command [ANSI 92] is as follows:

```
GRANT <privileges> ON <object>
    TO <users>
    [WITH GRANT OPTION]
```

The REVOKE command can be used to remove specific privileges from users that have previously been granted privileges. The CASCADE clause, if specified, recursively revokes privileges from users who have received the privilege from a user whose privilege has been removed. That is, the privileges are taken away from not only those users that are specified in the REVOKE statement, but also those users who were granted the privileges by the users specified in the REVOKE statement. The cascading of revocations will continue through the chain of users who were directly or indirectly granted the revoked privilege. The SQL syntax for REVOKE is similar to GRANT:

```
REVOKE <privileges> [ON <object>]
    FROM <users>
    [CASCADE]
```

These two commands enable the defined privileges to be managed by the owners of the objects in the database.

3.1.3 Use of VIEWS to Restrict Access

In addition to using the privileges described above to provide specific types of access to entire objects, there is also a mechanism within SQL to provide access to only certain portions of the contents of objects, or certain aggregate information about the contents of the object.[4] This is done through the use of views.

Views are basically canned (stored) queries. When a user creates a view, the SQL engine insures that the creator of the view has permission to access all the views or base tables referenced by the view before the view is stored in the database. Now the user can use this view to access the underlying data in the specified way. Note that this access checking is normally done only when the view is created, not when the view is used to access the underlying tables. This, in and of itself, does not provide any additional form of access control, since the user who created the view can access the underlying tables directly. However, what the user can now do is GRANT other users access to the view they created. This has the effect that these other users now have access to the information presented by the view, which they might not otherwise be able to access.

Views can restrict access in a number of ways:

- to specific columns, using SELECT or UPDATE,
- to specific rows, using the WHERE clause,
- to aggregates of the information accessed by the view, using the various aggregate functions supported by SQL (e.g., SUM, AVE, MIN, MAX).

For example, consider the following base table:

```
EMPLOYEE(EMP#, NAME, DEPT, SALARY)
```

This table contains, for each employee, his or her employee number, name, department name, and salary. Access may be granted to this entire table through a VIEW as follows:

```
CREATE VIEW V-EMP
        AS SELECT*
        FROM    EMPLOYEE
```

The “*’’ is a wild card that represents all columns within the table. A finer granularity of access control may be specified by restricting the user’s access to certain columns of the EMPLOYEE table as follows:

```
CREATE VIEW V-EMP-LIST
        AS SELECT EMP#, NAME, DEPT
        FROM EMPLOYEE
```

A user’s access may be further restricted to specific rows within the EMPLOYEE table. For example, a user’s access may be restricted to only those employees within the accounting department with the following view:

```
CREATE VIEW V-EMP-ACCT
        AS SELECT EMP#, NAME, DEPT
```

4. When comparing access controls applied at the granularity of an object with access controlled at the granularity of subsets of the objects contents, we will frequently refer to the entire object as a container.

```
FROM EMPLOYEE
WHERE DEPT = ACCOUNTING
```

A user's access may also be restricted to only aggregate information about the rows within the EMPLOYEE table. For example, rather than giving a user access to specific employee salary information, the user may be presented with only average salary information for each department with the following view:

```
CREATE VIEW V-DEPT-AVERAGE
    AS SELECT DEPT, AVG(SALARY)
    FROM EMPLOYEE
    GROUP BY DEPT
```

Using this ability to create views, and then assigning specific privileges for accessing these views, rather than granting direct access to the underlying views or base tables, a very rich set of DAC controls can be specified for a database.

Given the privileges described in Section 3.1.1 and the use of views to further restrict access to base tables, the DAC restrictions to database objects can be summarized as follows:

Object	Privilege	Granularity
Object Definition Ex: • Database • Table • View	• Create • Drop • Alter	N/A
Object Content Ex: • Table • View	• Insert • Delete • Select • Update • References • Index	• Container • Column • Row • Aggregate Functions
Stored Procedure	• Execute	N/A

Table 3.1: Summary of SQL DAC Capabilities

3.2 THE TYPICAL SQL ENGINE

As was stated at the end of Section 1.3, current DBMS architectures typically place the entire SQL engine inside the TCB since it is used to enforce DAC. Due to the size of the typical SQL engine, such an architecture will make it extremely difficult to meet the B2 or B3 assurance requirements. The approaches described in Section 5 describe how various parts of the SQL engine can be excluded from the TCB in order to achieve higher assurance in the resulting DAC mechanism. As background for the discussion in Section 5, this section describes the typical components of an SQL DBMS Engine and how these components normally support the enforcement of DAC.

The design of a SQL engine in practice is heavily influenced by the need to provide good performance. However, regardless of the actual design of the software, one can think in terms of certain functions the SQL engine must perform. We will use the components described below to provide a common framework for considering alternative security architectures. A SQL engine normally includes the following primary components:

- Parser (Accepts queries submitted to the DBMS, determines whether a submitted query is syntactically correct and translates the query into a form used by other DBMS components)
- Optimizer (Creates an execution plan which attempts to minimize the number of tuples that must be fetched in order to satisfy a given query)
- Executor (Executes the execution plan and returns results)

In addition, the following smaller components deal specifically with DAC control:

- Access rights checker (Determines whether permissions allow the requested operations to take place)
- Access rights grantor / revoker (Supports the GRANT/REVOKE commands)

Some form of interface is also provided to permit a user to submit requests to the SQL engine. However, this interface is typically outside of the SQL engine. An evaluation interpretation has determined that it is not required to use a Trusted Path for modifying or viewing DAC privileges, so even this aspect of the user interface can reside outside the TCB.[5] The engine itself is responsible for fielding user requests (e.g., SELECT, INSERT, UPDATE, DELETE) while the user interface is responsible for presenting this information to the user and facilitating generation of new requests of the SQL engine. These components are depicted in Figure 3.1.

The Parser, Optimizer, and Executor tend to be very large, complex pieces of software, the inclusion of which will typically prevent a TCB from meeting the B2 or B3 assurance requirements. On the other hand, when the components directly doing the maintenance and checking of access rights can be cleanly separated from other components, they are typically simpler and more suitable for inclusion in a high assurance TCB. One should note that this separation is likely to come at the cost of some performance degradation.

5. Interpretation C1-CI-01-86.

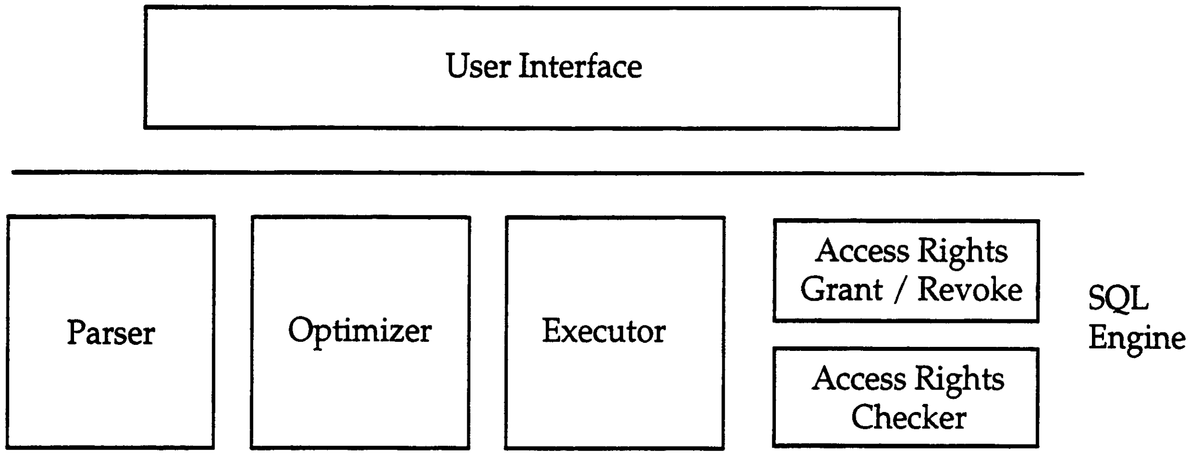

Figure 3.1: The Typical DBMS

3.3 TYPICAL DBMS ENFORCEMENT OF DAC

Given the SQL DAC capabilities described in Section 3.1 and the SQL Engine components identified in Section 3.2, this section describes how the typical DBMS enforces the SQL DAC capabilities. We first talk about access rights management, and then enforcement of access controls.

Access rights are managed through the use of the GRANT/REVOKE SQL commands. Since most DBMSs store access rights (privileges) in tables, the typical DBMS will use the SQL engine itself to help process any GRANT/REVOKE requests. This is similar to the fact that metadata (the data that defines the database, and its tables and views) is also stored in tables and accessed using the same functions provided for accessing the data stored in the database. This is done normally for convenience since the functions already exist, not out of necessity. However, this causes the bulk of the SQL engine to be part of the TCB in order to support access rights management.

Depending on the DBMS, the access rights checker can perform access control checks during various stages of a query. Typically, checks are done just after parse time to ensure that the user has the correct permissions to access the referenced database objects using the given commands. Checking can also be done after optimization time, or during each fetch. This access check is similar to the type of access check that would be done whenever a user requests access to a file in an OS. The DBMS must ensure that the user has the correct permission to access the object (e.g., SELECT access to the table, CREATE TABLE access to the database, etc.).

If the DAC policy is based solely on access permissions for named objects such as tables, the TCB could, in principle, include only the mechanism required to implement the GRANT/REVOKE commands and enforce access rights checking on the named objects. This would require separating out this functionality from the rest of the SQL Engine. The reason then for including the entire SQL Engine in the TCB is the convenience of using it to store and retrieve access rights data stored in tables, the performance advantages of an integrated SQL Engine, and the desire to reuse an existing SQL Engine without major modification. This results in the TCB as depicted in Figure 3.2.

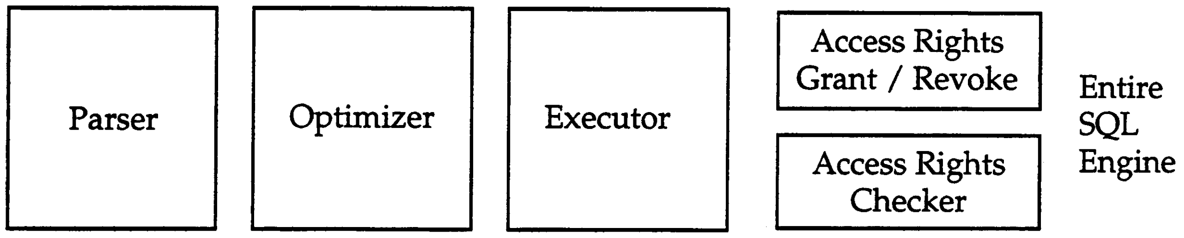

Figure 3.2: The Typical DBMS TCB

Organizations may well want to enforce a policy in which access rights depend on the contents of the database. This can be done by providing DBMS access controls based on views. However, to correctly enforce such a policy one must trust that the views are correctly interpreted. To provide a DBMS mechanism to enforce such a policy, one must trust that queries are correctly evaluated. View based DAC is discussed in detail in Section 4.2.

Now that we understand how DAC is typically done, we consider a few specific examples from evaluated DBMSs to see how they vary from implementation to implementation.

Trusted Oracle

The enforcement of DAC in Trusted Oracle7 is identical to that in Oracle7. Trusted Oracle7 only makes claims to the fact that they protect access to the contents of objects (e.g., base tables, views). This means that they protect access to view definitions, to ensure that only authorized users can access the view, but they make no claims about correctly interpreting the view, or that the filtering capability of views is even part of their DAC policy. This has been deemed acceptable for satisfying the TCSEC DAC requirements that all named objects be protected by DAC.

Informix-OnLine/Secure

Informix-OnLine/Secure's DAC enforcement is similar to that provided by Trusted Oracle7. They protect the contents of objects but do not make any claims as to the correctness of view interpretations, nor do they include the filtering capability of views in their DAC policy.

Achieving High Assurance DAC

As neither of these DBMSs was targeted above the BI level, they were not required to meet the architecture and minimality assurance requirements that higher assurance levels require. For those DBMSs that are targeted at these higher levels, the challenge is to determine ways to exclude parts of the SQL engine from the TCB while still providing adequate DAC functionality. Various approaches to achieving this goal are described in the Section 5.

SECTION 4

HIGH ASSURANCE DAC ISSUES

This section discusses several significant issues related to DAC policies and implementing high assurance DAC in DBMSs. Section 4.1 addresses weaknesses of DAC policies even when the DAC mechanism is correctly implemented. Section 4.2 discusses issues introduced by using views as a mechanism for enforcing DAC. Section 4.3 considers the issue of balanced assurance.

4.1 EFFECTIVENESS OF DAC POLICIES

DAC policies in information systems regulate the management, protection, and distribution of sensitive information based on a decision of the information owner. DAC policies can be used to implement organizational need-to-know policies in which access to information is controlled based on the information owner's determination that a user has **a** requirement to access the sensitive information in order to perform official tasks or services. Operating systems enforce DAC policies on operating system objects (e.g., files). DBMSs enforce DAC policies on DBMS objects (e.g., tuples, data elements) which are more granular (and may be stored in files). DBMSs also enforce DAC policies based on the content of the object (content-dependent DAC) or based on the context of the object (context-dependent DAC).

For example, a DAC policy defining access control on employee information may be implemented in an operating system by restricting a user's access to the employee file. A DBMS, however, may implement this same policy to a finer granularity. The DBMS may limit access to only selected attributes (e.g., all attributes but salary). The DBMS may also enforce a content-dependent policy based on data stored in the object (e.g., Joe has access to employee records for the engineering department) or a context-dependent policy based on the implicit relationships within the object (e.g., Joe has access to employee records for employees he supervises and those below him in the chain of command). Content and context dependent access control is described in more detail in Section 4.2.

In considering high assurance DAC, it is important to be clear about different aspects of assurance. The TCSEC considers the system access control policy and its model as being integral to the overall assurance that can be provided by an implementation of a trusted system. Other trusted system criteria, particularly the European Information Technology Security Evaluation Criteria (ITSEC) [ITSEC 91], consider *effectiveness assurance* as well as *correctness assurance* aspects of policy and its implementation. The ITSEC defines *effectiveness* as how well security is provided in the context of its actual or proposed operational use, while *correctness* is how accurately a component reflects its security requirements. It is noteworthy that the ITSEC's division between effectiveness and correctness places many implementation considerations for TCSEC B2+ requirements *(e.g.,* TCB minimality, structure, modularity, information handling, etc.) into the category of *correctness* rather than *effectiveness.* It is also noteworthy that ITSEC effectiveness requirements bring together policy and implementation considerations as part of the architectural abstraction. This appears to be sound reasoning, as it entertains the question, under effectiveness assurance, of whether there exists *any* possible implementation of the policy that would enforce the security objectives of the enterprise; while correctness assurance issues pertain directly to the properties of one specific implementation. With this context, one can consider more precisely how to interpret the term high assurance DAC in light of known limitations of DAC for controlling access to information.

It is well-known that discretionary access controls have fundamental limitations [NCSC 87]. These limitations are effectiveness issues. That is, no matter how much assurance is provided that the policy is correctly implemented (i.e., correctness assurance), the issues discussed below will apply as long as the policy permits propagation of access rights or replication or modification of persistent objects. This section discusses some of these limitations. The rest of this document then focuses on correctness assurance with the assumption that the chosen system DAC policy is sufficient to enforce important policy requirements of the using organization (i.e., effectiveness assurance requirements are fulfilled).

4.1.1 Propagation of Access Rights

MAC access rights are determined based on security attributes of subjects and objects, and these MAC security attributes are not changeable by standard users. Unlike MAC, in DAC the determination of who has access rights to an object and how (or whether) these rights may be given to others can be made by the owner of an object. Each object has an owner (the creator of the object), and the owner may decide to grant accesses to other users. Many different implementations of the discretionary access control policy exist, each giving various forms of access control on objects to the object owner. In many systems, the owner can decide to extend the authority to other users. Hence, access permissions can propagate through direct and indirect grant of permissions. Given this type of policy, one must consider whether or not it is decidable if there is a reachable state in which some subject may possess a particular access to an object (i.e., "the safety problem").[6] Since users can grant access rights to others under the DAC policy, safety has been shown to be undecidable by Harrison, Ruzzo, and Ullman [Harrison 76] for many access matrix models. A summary of the current status of the safety problem is found in [Sandhu 89].

Researchers have suggested new mechanisms designed to solve the problems with DAC. These efforts include:

- Propagated Access Control (PAC). Graubart proposed PAC lists (PACLs) as a mechanism for enforcing the ORiginator CONtrolled (ORCON) security policy [Graubart 89]. PAC shares some of the characteristics of both MAC and DAC.
- Owner-Retained Access Control (ORAC). McCollum proposes the ORAC mechanism to enforce identity-based access controls [McCollum 90]. With ORAC, ACLs propagate to new objects. The resulting ACL on the new object is the intersection of the ACLs on all objects from which the new object was created.

6.In the terminology of Harrison, Ruzzo, Ullman, a configuration of a protection system is "safe" with respect to a particular access right if there is no sequence of the commands permitted by the system which can add the right to a cell in the protection matrix where it was not found in the initial configuration. The safety problem refers to determination of whether a given configuration is safe for a particular right. [Sandhu 89] provides the following description in current terminology of the safety problem. "In its most basic form, the safety question [HRU] for access control asks: Is there a reachable state in which a particular subject possesses a particular privilege for a specific object?"

- Typed Access Matrix (TAM). Sandhu suggests a new access matrix model, the TAM, which introduces strong typing into the Harrison, Ruzzo, and Ullman model [Sandhu 92].

- Generalized Framework for Access Control (GFAC). Abrams proposed GFAC as a framework to go beyond the traditional thinking of MAC and DAC and formulate a unified general access control policy [Abrams 91].

4.1.2 Trojan Horses

Trojan horses pose the most serious threat in systems that enforce only a DAC policy. The Trojan horse problem arises from the difference between a user's intentions and the user's privileges. Any program executed on behalf of the user executes with that user's privilege, but may not match the user's intentions. This program may violate the user's intentions through its use of the DAC mechanisms in several ways.

One way for a Trojan horse to violate the user's intentions is through direct manipulation of the object's privileges. For example, suppose it is known that Tom uses a DBMS application supplied to him by Dick (or some other malicious user). Dick could have embedded a small piece of code (i.e., the Trojan horse) into the application. When Tom executes the application, it is given Tom's DAC privileges. Assume that Tom has GRANT with the WITH GRANT OPTION access to the Employee relation. In this case, if Tom executes the application with a Trojan horse, the Trojan horse code will now have the ability to GRANT other user's access privileges to the original Employee relation. This example illustrates the risk that is taken when giving GRANT with the WITH GRANT OPTION access to any object. While this risk cannot be prevented, it can be detected through careful auditing and audit analysis.

Another way for a Trojan horse to violate the user's intentions is through copying the information into another object and then grant additional access to the copied file. This time assume Tom does not have GRANT with the WITH GRANT OPTION access to the Employee relation, only READ access. Because the application with the Trojan horse is executing with Tom's privileges, the Trojan horse can read the EMPLOYEE relation and write a copy into another relation called the *COPY-OF-EMPLOYEE* that gives Dick read access. The copying takes place without Tom's knowledge and Dick now has read access.

Additional restrictions may be placed on Tom's access to the EMPLOYEE relation in an attempt to limit the Trojan horse effectiveness. However, as long as Tom retains access to the EMPLOYEE file and the ability to share information with other users (i.e., as long as the system remains useful), the DAC mechanisms will remain vulnerable to Trojan horse attacks. For example, if Tom was restricted from copying the EMPLOYEE relation, the Trojan horse could relay the information within the EMPLOYEE relation to Dick via a mail message. MAC was specifically designed to prohibit embedded Trojan horses from leaking information from objects at higher security levels to objects at lower security levels.

4.2 DBMS DAC POLICIES

Given the variety of SQL DAC privileges and the expressive power of views, a wide variety of DAC policies can be enforced in a DBMS. Enforcing these policies with high assurance has a number of implications which are discussed in the subsections below.

DBMS DAC policies are generally divided into the following four categories:[7]

- Content Independent DAC
- Content Dependent DAC
- Context Independent DAC
- Context Dependent DAC

Content Independent DAC refers to the ability to control access to an object regardless of the contents of the object (i.e., only by name). This includes all the privileges that might be granted to the object (e.g., insert, select, update, delete) as well as the ability to select on particular columns of a table, as long as aggregate functions are not used. Given this, the query results are independent of the values of specific attributes contained within a table. This type of DAC can be provided in standard SQL.

Content Dependent DAC refers to the ability to control access to an object based on the contents of the object. This includes using the WHERE clause to check for particular values or relationships between values, as well as aggregate functions used in a SELECT statement. Using content dependent DAC, query results will depend on the values of the various attributes contained within a table. This type of DAC can be provided in standard SQL through views.

Context Independent DAC refers to DAC that is not affected by the context of the request. Context can include: time, date, day of week, terminal id, previous tuples accessed, etc. This type of DAC can be provided in standard SQL.

Context Dependent DAC refers to DAC that takes into consideration the context of the request. Depending on the type of context being considered, a highly sophisticated mechanism might be needed to keep track of this information. For example, if the previous tuples accessed are considered part of the context, then some type of history mechanism must keep track of this information. However, other types of context, such as time, date, day of week, or terminal ID should be easy to provide. The specific context that can be considered is dependent on the features provided by the DBMS.

Although enforcement of content dependent DAC is not required for TCSEC evaluation, organizations may well want to enforce a policy in which access rights depend on the contents of the database. This can be done by providing DBMS access controls based on views. However, to correctly enforce a view-based DAC policy, one must trust that the views are correctly interpreted. This raises several issues with respect to enforcing view-based DAC with high assurance.

4.2.1 Complexity of View Mechanism

One important issue is the complexity involved in correctly interpreting views. A view is computed based on the contents of the underlying database. This might appear to be similar to interpreting an operating system object defined by bits that are part of the contents of a disk. However, there are obvious differences here:

7.For more discussion on content and context based access controls see [Marks 94].

(a) the view is computed from objects that are directly accessible by users, whereas (with proper encapsulation) neither the operating system objects used to store the database nor the disk sectors used to store the files should be;

(b) views are user defined as opposed to being defined by the system; and

(c) the computation of view instances is typically much more complex than the other instances.

If the DAC policy is based on restricting user access only to data accessible through specific views, it is necessary to show that the "right" data is returned as the result of executing a view. To believe the correct data is returned, one must trust the mechanism computing the view. Because of the complexity of this mechanism for general views, it will be hard to achieve high assurance for a general view-based DAC mechanism.

Complexity of the mechanism required to enforce the system DAC policy is a central issue for high assurance DAC. As noted in Section 2.1.2, the TCSEC introduces several assurance requirements at B2 which constrain the potential system architectures which may be used. If a large complex mechanism such as a SQL compiler must be included in the TCB these requirements can be extremely difficult to meet. The introduction of additional assurance requirements at B3 makes it virtually impossible for such a complex mechanism to meet these requirements.

For enforcement of DAC controls on tables or databases, the complexity of DAC enforcement will be similar to DAC enforcement in an OS. However, if views are used for access control, a much more complex mechanism may be required. The most difficult problem encountered with views is the size of the TCB. It tends to be very large if the view mechanism is implemented the same way in high assurance DBMSs as in untrusted DBMSs because almost the entire DBMS needs to reside in the TCB. All the components of the DBMS involved in computing the view need to be considered as a part of the TCB responsible for enforcing DAC. For this TCB, there must be sufficient assurance that the correct data values were returned to the user in concert with the discretionary constraints defined in the view.

To overcome this difficulty, researchers have suggested various approaches for defining high assurance DAC. Most of these approaches revolve around simplifying the view mechanism. These approaches place restrictions on the types of views that can be generated in order to simplify the TCB. These approaches are discussed in Section 5.

4.2.2 Disjoint vs. Overlapping Objects

Another issue associated with enforcing DAC through views is the fact that views can overlap. Base tables *are* disjoint by definition. Views are not only *not* disjoint in general, but it can be mathematically hard to determine by inspection if their instances do or do not intersect. Further, a pair of view instances may be disjoint at time t_i and they may intersect (or fully coincide) at time t_{i+1}.

However, it is clear that at *fetch* time (at a very low level of abstraction in the DBMS engine) it is possible to compare individual attributes in individual tuples of a base table. This is because the low-level engine is capable of identifying each attribute of a tuple in terms of its data type (extracted from the schema) and from implementation detail about its displacement and extent from the base address of the tuple.

Thus, if a view is defined on a single table and includes the primary key, it is possible for the DAC enforcement layer of the TCB to evaluate projections of expressions on table ***R*** of the form

$$R.attr_i \ \textbf{<relOp>}\ \boldsymbol{R.}attr_j$$

that correspond to view WHERE clause expressions extracted from the definition of view *V* of the form

$$V.attr_i \ \textbf{<relOp>}\ V.attr_j$$

where **<relOp>** is any of the standard comparison operators *(e.g.,* $\leq < = \neq > \geq \in \notin \neg$). This enforcement can be performed directly from the schema and from definitions of the view that could be accessible in a canonical form to the DBMS TCB in an association with named users. The DAC view instantiation mechanism would not need to rely on or be based on the SQL compiler or any of its supporting mechanisms. Indeed, even were the query to be modified by the untrusted parser, compiler, optimizer, or SQL engine, the test could still be applied to ensure that the only data extracted would be data to which the user possessed a valid SELECT, DELETE, UPDATE, or INSERT right.

4.2.3 DBMS Functionality and the Safety Problem

The complexity of view based DAC policies can also make it difficult to determine what data a user can access since the data may be accessed directly or through a view. Access through views is preferred in many real applications over granting users direct access to the base relations. Access through views is a means of ensuring that the conditions of access or update to the database are controlled in ways that relate to the content of the 'relevant' and 'important' elements and relationships within the database. These elements and relationships are determined by a person assigned administrative duties over all or a portion of the database. Consequently, it is expected that users will, in general, be able to query or update data through a view that they cannot access directly in the base relations — indeed, SELECT or UPDATE access to the base tables may be explicitly prohibited to these users. As a result, databases present even more complexities than operating systems for determining which users can ultimately gain access to particular data.

In some cases, analysis may show that an individual user is on an exclusionary list for selecting on a specific base relation, while the user has select access to one or more views that permit select and update on a subset of a base relation or set of base relations. Since different named views may overlap, access to several named views may together result in the user being able to select or update on the *entire* relation. The above view properties hold independent of the means by which views and their interpretation/enforcement have been implemented in the DBMS architecture.

This complexity gives richness to the meaning of DAC and its relationship to a provable 'safety' property. In a container-based system, presence of an ACL is taken to mean that a user may obtain *direct* access to a container object if the (userID, mode) pair appears on the container object's ACL.

However, the introduction of content-based addressing characteristics of a DBMS leads to the possibility of an individual user having numerous means of addressing subsets, *S* of a database. If each of these means is achieved through views available to a user, then to claim that a user cannot *directly* view a given subset of a table is to claim that the user does not have access to any view whose defining WHERE clause identifies all or part of *S*. It is a difficult problem to determine whether this is the case. Hence, the strongest claim one could make about safety for DBMS access is probably of the form: a user may SELECT (or UPDATE) a portion *S* of a base relation if the user is authorized to some set of views *V* whose WHERE predicate evaluates to a set that includes *S* and the user has SELECT (or UPDATE) rights to *V*. This property probably does not have an only-if counterpart.

Note that this characterization of *V* and *S* is not very useful, and the restriction is not very satisfying. This property is independent of layering or modularity. It is much closer to trying to say something about a dynamically extensible TCB, since each database administrator may add a new view to the system at any time, and this new view immediately changes the access properties of the set of users who have access to the new view. It is a mathematically hard problem to identify the range of each view, even though the domain and the selection predicate are both well-defined.

4.3 BALANCED ASSURANCE

The original notion of balanced assurance was introduced by the SeaView project [Lunt 88] extending ideas in [Schaefer 84] and [Shockley 88]. It provided "a methodology for achieving a high level (Class A1) of assurance for a system as a whole by applying high assurance techniques to the part of the system enforcing the MAC policy while requiring assurance measures equivalent to C2 for those parts of the TCB enforcing non-mandatory access control policy." [Irvine 92]

Many proponents of balanced assurance have based their argument on the basic fact that any implementation in which the DAC enforcement mechanism *is fully* constrained by the underlying MAC enforcement mechanism cannot, by its very nature, violate the MAC policy and lead to a compromise of the MAC policy's safety properties. This argument can readily be shown to be sound relative to *all* of the system's lattice-based confinement and non-interference properties: if the DAC enforcement mechanism is fully constrained by an adequate MAC enforcement mechanism, neither overt nor covert channels can be exercised by the DAC component in a B2 or higher security architecture.

It is clear that an ineffective policy can be implemented soundly (correctly and in a form that can be assessed by a certifier or evaluator). It is also clear that an effective policy can be implemented unsoundly (with errors, poor structure, no information hiding, unsound coding discipline, etc.). Many balanced assurance advocates have taken the position that, because of the lack of a DAC safety property as described in Section 4.1, no correctness assurance requirements beyond those of C2 should be levied on the portion of a TCB that enforces DAC.

There are actually two distinct aspects that are embodied in these descriptions of balanced assurance: giving a class rating to the composed system TCB and providing the appropriate level of assurance for each TCB subset.

The first aspect is that a TCB composed of a Class A1 TCB enforcing a MAC policy and a Class C2 TCB (in particular a DBMS) should be considered a Class A1 system TCB. In essence, this is a naming issue. What does one call a C2 DBMS on an Al operating system? Given that the SeaView project was required to produce an Al DBMS, the project team proposed calling the composed TCB Al. The TDI did not embrace balanced assurance, concluding for hierarchical composition of TCB subsets that the composed TCB could not receive a rating [Tinto 92]. Note that this is different than partitioned composition used for networks where the composed TCB does receive a rating.

The second aspect is that regardless of the TCSEC Class associated with the system TCB, the degree of (correctness) assurance needed for trusted components should be proportionate to the security risks the component poses [Sterne 94]. In particular, one needs high assurance for a TCB that enforces a MAC policy, but may not need a similar level of assurance for other TCB subsets that enforce a more restrictive policy. The TDI fully supports this aspect of balanced assurance. Although this document concentrates on how to provide high correctness assurance for DAC mechanisms for those environments where such assurance is required, there are many environments where it is sufficient to provide a lower degree of assurance to a DAC mechanism which is constrained by a more highly assured underlying MAC mechanism. SeaView is described in the next section as an example of an approach based on balanced assurance.

SECTION 5

APPROACHES TO IMPLEMENTING HIGH ASSURANCE DAC

A number of different approaches can be used to implement DAC in a high assurance DBMS, reflecting divergent perspectives on the degree of high assurance and the meaning and uses of DAC within a multilevel secure (MLS) environment. Each perspective has its strengths and weaknesses, and the correct choice of approach depends on the requirements of specific applications. This section describes the various approaches suggested by different researchers for implementing DAC in high-assurance DBMSs.

This section first provides a brief description of using ACLs on database objects including their use in a prototype DBMS. It then presents a method for providing a high assurance rights management mechanism (i.e., GRANT/REVOKE) in Section 5.2. The remainder of this section addresses enforcement of these rights. Section 5.3 describes the SeaView approach that is based on balance assurance. Section 5.4 discusses the ASD Views research effort from TRW. Section 5.5 describes another approach suggested by Wilson of TRW for restricting views in order to achieve higher assurance. Section 5.6 describes a framework for reasoning about different DAC mechanisms including details of *pviews, qviews,* and *uviews* and a discussion of which of these mechanisms can be implemented with high assurance.

5.1 ACCESS CONTROL LISTS

The most direct approach in implementing DAC in a DBMS is to use an Access Control List (ACL) (or equivalent) mechanism to enforce access to DBMS objects. In theory, this approach can be implemented at any level of granularity (i.e., database, table, view, row, column, or element) within the DBMS. However, complexity and performance considerations dictate a limit to the applicability of the ACL mechanism.

At the database-level granularity, an ACL is associated with the entire database. The ACL contains a list of all the subjects that are allowed to access the database together with the types of access rights (e.g., create session, create table) of these subjects to the database. The entire database can easily be stored in one or more operating system (OS) files, and then the OS DAC mechanism could be used directly to restrict accesses to the database. This removes the need for a separate DAC mechanism for the DBMS. Applying DAC only at the database-level, however, is restrictive and loses most of the flexibility of discretionary access controls.

For table-level DAC, an ACL is associated with each relation of the database, giving the names of the users or groups allowed access to the table. Access rights that are appropriate to the table level include basic data manipulation functions (e.g., select, insert), the definition or modification of active data functions (e.g., triggers and stored procedures), view definition functions, and functions that allow the modification of the relation definition (e.g., create index, drop index). To be granted table-level DAC, users normally must first be granted access at the database level. Like database-level DAC, table-level DAC is easy to implement. Though table-level DAC is more flexible as compared to database-level DAC, it is still restrictive to only provide DAC for the database as a whole and for tables.

ACLs can also be associated with each view. Such an ACL specifies the users allowed to use the view. A view can restrict the information that a user is allowed to access. Access rights that are appropriate at the view-level include those appropriate at the table-level (e.g., select, insert, stored procedures, create index). The power of the SQL language in specifying view-definitions makes view-level DAC an excellent level of granularity for controlling access to DBMS objects. (Note that using ACLs only constrains access to the view definition. This does not imply anything about the correctness of the view interpretation.)

At least one commercial DBMS vendor implements an ACL mechanism in addition to SQL to provide access control to DBMS objects. Trusted Rubix controls access to databases, tables, and views via an ACL mechanism. Database ACLs support browse, modify, lookup, and grant privileges. Trusted Rubix tables and views support delete, insert, and grant privileges for whole tables and reference, select, and update privileges for columns.

The ACL mechanism could, in theory, be applied to DBMS objects of finer granularity. However, these implementations become difficult to manage and require large storage overhead and are therefore less commercially viable.

ACL mechanisms which protect database objects are of similar complexity to ACL mechanisms found in operating systems. One additional complexity is the use of GRANT/REVOKE to alter DAC privileges. The next section shows how GRANT/REVOKE can be handled with a simple mechanism suitable for a high assurance implementation. Since ACL mechanisms have been implemented in high assurance operating systems, this type of mechanism can be used to provide high assurance DAC for DBMSs. As noted above, however, the DAC policy implemented by using ACLs on view definitions does not of itself control the actual data content provided by executing a view. While content control is not required for evaluation, it may be required for enforcing the security policies of some organizations. How to provide this type of content control through views is discussed in Sections 5.4-5.6.

5.2 HIGH ASSURANCE GRANT/REVOKE

As indicated in Section 3, the typical DBMS stores access rights in tables and uses the SQL engine to access these tables. The use of the SQL engine is mostly a convenience, rather than a requirement. Since the SQL engine already exists, it is a natural mechanism to provide access to the data stored in the access rights tables. This then avoids the need to develop separate code for managing these tables. This is fine for providing the required functionality, but flies in the face of the assurance requirements because an SQL engine is large and complex. Typical implementations lack the structure required to separate out a subset of the SQL engine which is sufficient to support GRANT/REVOKE. Thus, the first step in providing high assurance GRANT/REVOKE is to develop a separate mechanism for storing and manipulating the access rights stored in the database.

To provide high assurance, the mechanism must be as simple as possible. System R used the following algorithm to enforce GRANT/REVOKE along with storing all the access rights in a single table [Griffiths 76; Fagin 78]. This algorithm is simple enough that it should be relatively straightforward to implement in a separate high assurance mechanism that is independent of the SQL engine. The algorithm works as follows:

Users may grant access only if they are the owner of an object or have been granted access with grant option. Users may only grant privileges that they hold. All grants must list explicit modes that are being passed and the system time stamps all grants and records the id of the grantor. This culminates in the following attributes being recorded for each grant: subject to whom granted, privilege being granted, table to which authorization refers, timestamp, userid of grantor, and with/without grant option. These attributes are respectively represented by the following symbols: s, p, t, ts, g, go.

Users who grant may also revoke. In System R, revocation was automatically recursive or cascading. The result of user g revoking p on table t from user s is defined to be the same as if all the authorizations for p on t granted by g to s had never been granted. Revocation was performed as follows: suppose user g revokes privilege p from user s. If there is no authorization for the privilege on the table granted by g to s, then the revoke is ignored. Otherwise, every authorization for privilege p on table t granted by g to s is deleted. If at least one of the deleted authorizations was with the with grant option, then the authorizations granted by s may also need to be revoked. Let ts be the minimum timestamp of S's remaining authorizations for privilege p on table t with the with grant option (if none still appears, then let ts = infinity). Then, each grant by s of privilege p on table t with a timestamp smaller than ts is deleted from the authorization table. This procedure is repeated for each user from which any authorization is revoked.

Given that this algorithm is fairly simple, it should be relatively straightforward to implement a small high assurance mechanism which implements this algorithm and then directly accesses the access rights table. Since access is only being performed to a single table, and no complex searches need to be performed in order to determine where an entry in the table lies, the update mechanism should be simple as well. If this or a similar approach is used, then the GRANT/REVOKE aspect of the DAC mechanism can easily be implemented with high assurance.

5.3 SEAVIEW

The SeaView project developed research results in many areas of MLS DBMS policy design and implementation. Of particular interest for DAC is the approach SeaView took to implementing DAC in a high assurance MLS DBMS. SeaView relies on the underlying operating system TCB to enforce MAC as depicted in Figure 5.1. Data of different levels is stored in operating system containers of the appropriate level by decomposing multilevel relations into multiple single level relations. SeaView can then provide a DBMS instance at the level of a subject wishing to access the database, and this DBMS instance can run without the privilege to override the operating system MAC controls.

The DBMS is untrusted with respect to MAC. The assurance that the MAC policy is correctly enforced is exactly the assurance one has in the operating system TCB. This approach would typically utilize a high assurance operating system as the platform for the DBMS. Since the DBMS cannot cause a violation of the MAC policy, the SeaView developers argued that less assurance was required for the trusted portions of the DBMS. Those trusted portions of the DBMS include the mechanism enforcing DAC controls to data within the database. This approach does not provide high (correctness) assurance DAC. However, it does provide DAC within a system for which there is a high degree of assurance that the MAC policy is correctly enforced. This is precisely the balanced assurance approach. While the SeaView architecture could be used in conjunction with a high assurance DAC mechanism, one of its main advantages is the ability to provide high assurance MAC with the use of a lower assurance single level DBMS.

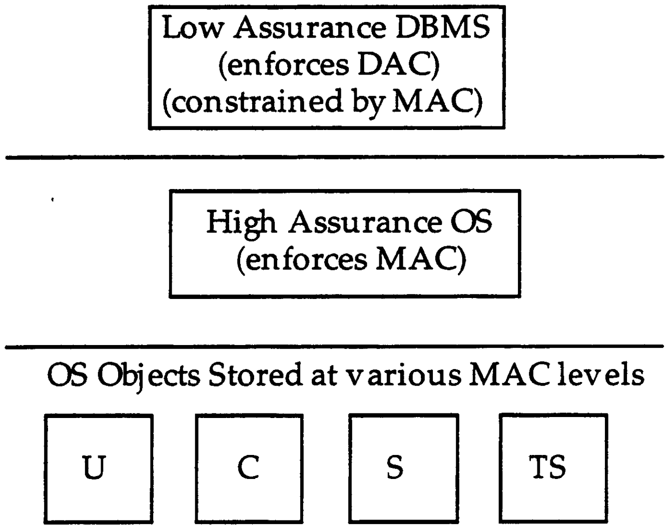

Figure 5.1: General SeaView Approach

The SeaView architecture implements the extensible TCB concepts proposed by Schaefer and 5chell. Shockley and Schell refined these concepts and introduced the term *TCB subset* along with what they called *incremental evaluation* for hierarchical subsets [Shockley 88]. Much of the TDI is based on this Shockley and Shell paper.

5.4 ASD VIEWS

The ASD_Views project was a conceptual design for a high assurance MLS DBMS with views as the object for MAC and DAC [Garvey 88]. To limit the size of the TCB, this approach restricts the generality permitted in the definitions of secure views to a fraction of the full power of the relational calculus, thus minimizing the code needed to ensure the correct execution of secure views. To restrict the size of the query language available to define a secure view, views can be defined to use a subset of the relational algebra. Views can only be defined by the System Security Officer (SSO) using a trusted path. Users can only access data through views. Whether and how a user can access a particular view is determined by the access list for the view. The SSO can add or delete permissions for a user or a group via GRANT and REVOKE commands. Permission to grant or revoke access cannot be given to any user other than the SSO. Specific permissions to a view can be explicitly denied via a “deny” command. In the event there is a conflict in permissions between permission granted or denied via user name and permission granted or denied via group membership, the permissions explicitly associated with the individual take precedence. It is worth noting that although the functionality available in the view definition in ASD_Views is restricted, the flexibility is still greater than in a DBMS that allows only rows or columns in a relation as security objects. The view definition for ASD_Views only allows a set of rows or columns from a single underlying base table to be specified through simple selection and projection operations. Joins, aggregate functions, and arithmetic expressions are not allowed.

The user or application program interacts with an untrusted SQL processor which provides a full SQL interface built on top of the secure views processing done in the TCB. The code for processing the query resides outside the TCB. The query processor decomposes the query into requests to read rows from secure views defined by the TCB. Thus, all query processing is done over the secure views implemented within the TCB. This would result in the following TCB:

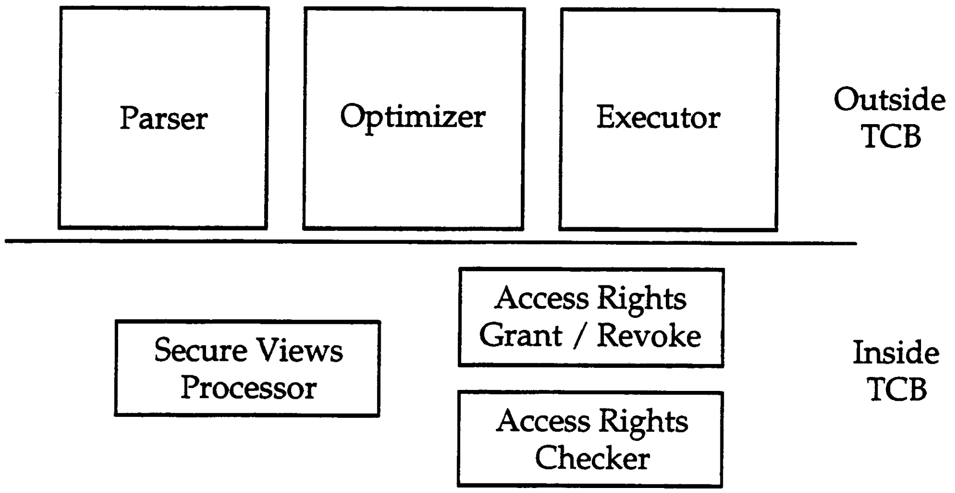

Figure 5.2: Proposed ASD_Views TCB

5.5 RESTRICTED VIEWS

Wilson, also of TRW, proposed an approach that would provide the full generality of SQL by assuring the DBMS is trusted to be correct, i.e., to execute queries correctly [Wilson 88]. This approach partitions the DBMS into a ‘‘non-malicious’’ part that searches the database and a trusted part that verifies that the data found satisfies the user’s query. The trusted part is an SQL interpreter with the remainder of the code being untrusted. This approach promises that every tuple returned to the user satisfies the search criteria because a post-check is performed on the tuples returned to ensure that they meet the search criteria. The code in the TCB establishes the search criteria and checks the results against the criteria. If it can be proved that the returned results meet the criteria, then the rest of the code can be placed outside the TCB. This results checking code must be non-bypassable, thus ensuring that if the other code makes a mistake, unauthorized data will not be returned to the user.

This implementation approach, shown in Figure 5.3, is as follows: a trusted process receives queries from the user terminal or application process. It expands the query to include definitions of any views being accessed, and passes the expanded query both to an unmalicious database searcher and to a relatively small, trusted query interpreter. The unmalicious searcher searches for the tuples in the database that satisfy the expanded query. The purpose of this code is to find the tuples as fast as possible. It also performs query compilation and optimization, index management, and other functions involved in searching for tuples. It passes the tuples to the trusted interpreter, which verifies that they indeed satisfy the expanded query. Although the trusted interpreter might be relatively slow, performance should not be greatly affected because the interpreter normally works on only a fraction of the tuples that would have to be accessed by the database searcher.

This approach is similar to the Integrity Lock approach for MLS DBMSs [Knode 89]. The basic idea is to let the complex SQL engine perform its search and retrieval functions as effectively and efficiently as possible outside the TCB and then verify that the correct results are returned to the user based on the search criteria. This approach then has the same potential flaws as the integrity lock approach in that the untrusted SQL engine could signal whatever information it desires to the user. Wilson discusses some ways to control this covert channel [Wilson 88]. However, because of the channel, it is unlikely that this approach would suffice for a high assurance DBMS using views as the object for MAC. However, a hybrid using this approach for view-based DAC together with traditional enforcement of MAC on labeled objects such as tuples may be possible.

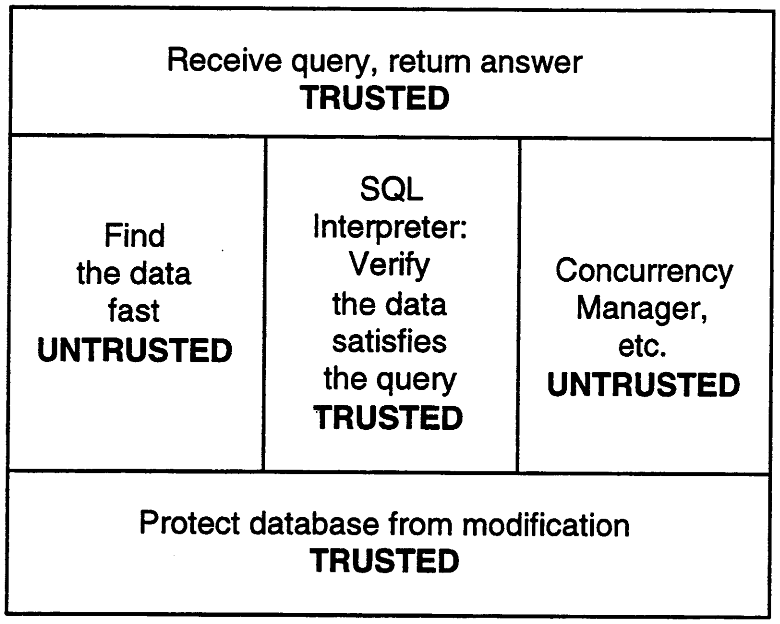
Receive query, return answer
TRUSTED
Find the data fast
UNTRUSTED
SQL Interpreter: Verify the data satisfies the query
TRUSTED
Concurrency Manager, etc.
UNTRUSTED
Protect database from modification
TRUSTED

Figure 5.3: Assured "Correct" Architecture

5.6 U-, P-, AND QVIEWS

This section describes Schaefer's approach to defining three degrees of assured DAC policy [Schaefer 94]. These are Uninterpreted DAC Views (*u*views), Interpreted Assured Primitive Views (*p*views), and Interpreted Qualified Views (*q*views). Each of these is described below.

5.6.1 Uninterpreted DAC Views (*u*views)

The Uninterpreted DAC Views *(uviews)* approach can be used to provide high assurance DAC to named objects, specifically: the database, metadata, base tables, view definitions, and stored programs. This is done through the use of ACLs associated with each of these objects. High assurance can be provided in this case at the same level of assurance that can be provided by an OS using ACLs. The entire SQL engine can be excluded from the TCB as it is not needed to enforce the access control policy to these objects. The SQL engine can request access to these objects on a user's behalf in a manner similar to applications running on an operating system. This is shown in

Figure 5.4. More information on implementation approaches for *uviews* is available in [Schaefer 94].

However, as the name implies, the security policy associated with *uviews* does not make any claims as to the correct interpretation of views. Thus, this approach provides a high degree of assurance that only authorized users will be able to access a view definition, but it makes no claims that, once accessed, the view will be interpreted correctly by the SQL engine. This method treats view definitions as just another object to be protected, rather than the stored program that it represents.

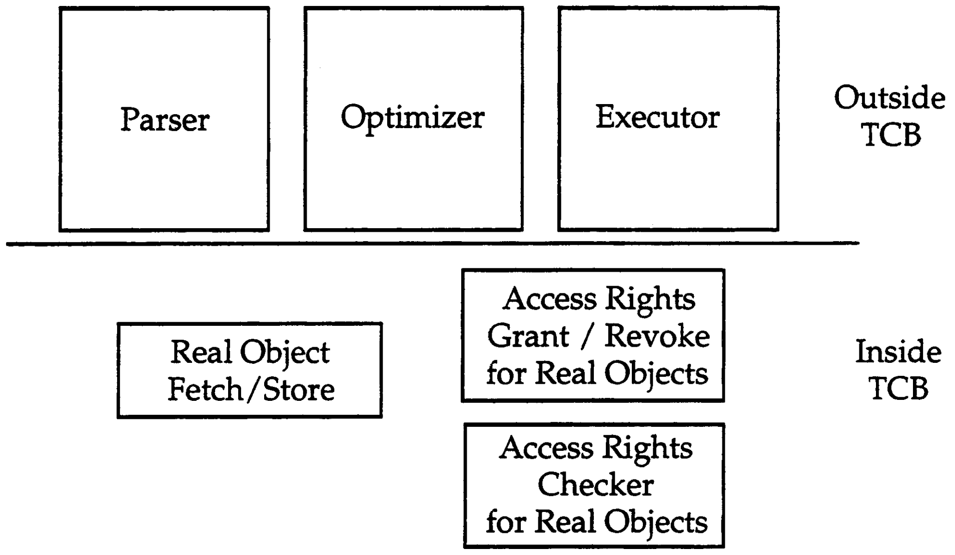
Parser
Optimizer
Executor
Outside TCB
Real Object Fetch/Store
Access Rights Grant / Revoke for Real Objects
Access Rights Checker for Real Objects
Inside TCB

Figure 5.4: uviews TCB

The policy described here is the same basic approach that has been used by both Oracle and Informix in their evaluated products. They both protect all objects as containers of information, but they do not make any assurance claims as to the correct interpretation of defined views. The approach they have taken is capable of achieving high assurance, but they have not yet done so since they were both targeted (and are currently evaluated) at B1. This approach is exactly the same as the ACL approach described in Section 5.1.

5.6.2 Interpreted Assured Primitive Views (*pviews*)

The Interpreted Assured Primitive Views *(pviews)* approach extends *uviews* to include the virtual objects formed from Primitive View Definitions and their manipulation. The primitive view is a simplification of the standard SQL view abstraction. Each named *pview* is defined to relate to precisely one base table. *pviews* are built from the base table by selecting a subset of the tuples and “projecting out” some of the attributes. The determination of whether or not to include a particular tuple is based on comparisons of the values of an attribute with other attributes or a scalar value. In a notation borrowed from Query By Example [Zloof 77] with modified semantics to fit the needs, a *pview* definition can be expressed as shown below:

Key	A_1	A_2	A_3		A_n
+	$\pm \mathbf{r}_i\, e_1$	$\pm \mathbf{r}_j\, e_2$	$\pm \mathbf{r}_k\, e_3$	...	$\pm \mathbf{r}_l\, e_n$

where $e_i \in \{A_i\} \cup$ Scalar Constants and **r** is a standard comparison operator. The ± indicates whether the attribute value will be included in the exported tuple instance, if -, attribute A_i is "projected out" *(i.e.,* not selected) and replaced by a standard default value for A_i *(e.g.,* ***null****).* A + indicates the attribute is to be included. The rule then for deterring whether a given tuple will be included in the *pview* instantiation is:

If every instance of a non vacuous expression $\pm\, \mathbf{r}_i\; e_1$ is satisfied for the tuple, then the tuple is included in the view and for each attribute A_i, if +, attribute A_i is included in the exported tuple instance, if -, attribute A_i is replaced by a standard default value for A_i in the exported tuple instance.

The *pview* instantiation can thereby be expressed as:

Key	a_1	a_2	a_3	...	a_n

where each *a*- represents the value Of A_i or its default value as derived from the application of the *pview* definition.

It is possible that algebraic expressions in terms of the A_i and constants would be practical, but this begins to make the checker fairly complex. The exported tuple instance is passed out of the TCB boundary and presented to the untrusted SQL engine for subsequent compiler-dependent processing. As shown in Figure 5.5 only the simpler *pviews* interpretation needs to be done in the TCB. Schaefer provides a more detailed discussion on *pviews* implementation [Schaefer 94].

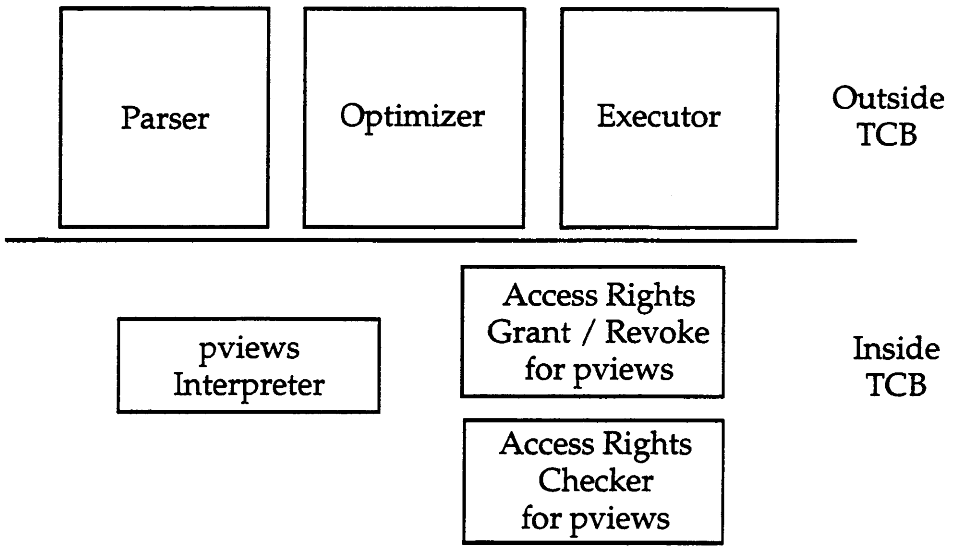
Parser
Optimizer
Executor
Outside TCB
pviews Interpreter
Access Rights Grant / Revoke for pviews
Inside TCB
Access Rights Checker for pviews

Figure 5.5: pviews TCB

5.6.3 Interpreted Qualified Views *(qviews)*

The Interpreted Qualified Views *(qviews)* approach extends *pviews* by generalizing the set of virtual objects which are used for access control to all those that can be generated using standard SQL. Given the expressive power of SQL (or an equivalent query language) *qviews* are thereby of far greater potential complexity than are *pviews.* In this approach, the claims made for the DAC mechanism are much greater than *uviews* where no claim is made of correct interpretation of views and greater than *pviews* where the claim of correct interpretation is made only for a restricted set of views. Because of the complexity required to generate this broader family of views, implementations of this policy can only be done with low assurance.

The *qviews* approach is functionally identical to the *uviews* approach. The only distinction is that *qviews* makes a claim as to the correctness of view interpretations, while *uviews* does not, which is purely an assurance issue. Although the overall *qview* approach can be done only with low assurance, certain aspects of the mechanism can be achieved with high assurance. The correctness of the DAC checks on DBMS objects can be assured to the same level as is DAC to traditional operating system named objects. A view definition V can be implemented to a high degree of assurance to ensure that the creator of V possesses the appropriate mode of access to each real named object referenced in V.

The correctness requirements for showing that *qviews* are enforced as part of the DAC policy would require showing that *qview* instantiations are produced correctly from their view definitions. High assurance can be achieved for the class of *qview* instantiations that are identical to *pview* instantiations *(i.e.,* when they correspond fully to disjoint *virtual* objects). However, the far greater complexity of the general mechanism would necessitate including the SQL compiler, optimizer, and engine completely within the TCB boundary as shown in Figure 5.6. It is presently beyond the state of the art to argue formally or through deep code analysis for the correctness of a mechanism of this size. In addition to the correctness of these mechanisms, additional assurance requirements for modularity, least privilege, TCB minimality, and information hiding would make it unfeasible to satisfy the TCSEC's security architecture correctness assurance requirements for B2, B3 and A1. For those reasons, *qviews* are still considered to be a low assurance DAC policy.

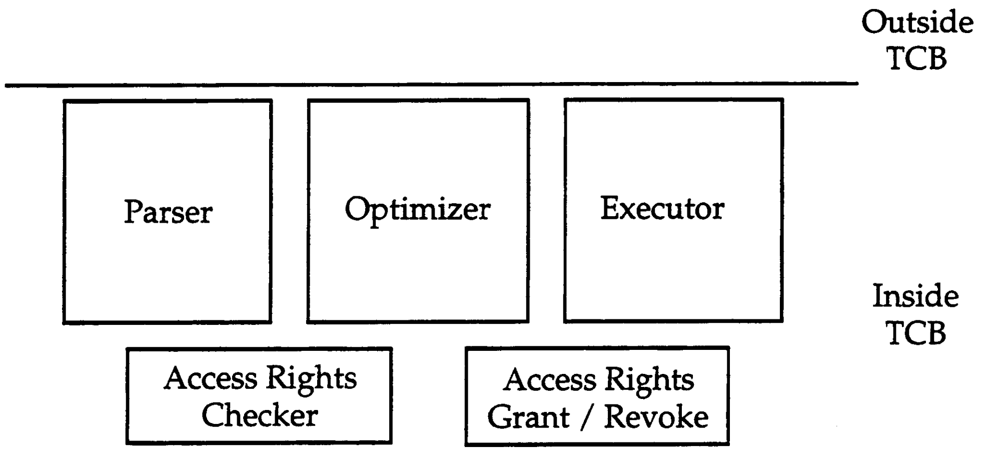

Figure 5.6: qviews TCB

SECTION 6

SUMMARY

This document has addressed the critical problems associated with providing high assurance DAC for DBMSs and discussed potential solutions to some of these problems. The difficult problem for DBMSs meeting higher level TCSEC requirements is the need to meet the assurance requirements. This is made even more difficult when it is desired to enforce a DAC policy with flexibility of content dependent access control. In general, the flexibility of the DAC policy chosen is inversely related to the degree of assurance which can be provided in the correct implementation of a mechanism enforcing the policy.

In order to put the discussion of DAC into context we have considered how effective DAC policies are even when their correct implementation is highly assured. In particular, the safety problem and Trojan horse problems mean that even a correctly implemented DAC mechanism cannot guarantee that a user cannot obtain data which the user was not supposed to be able to see. Given these characteristics of standard DAC policies one must consider the issue of balanced assurance. That is, one must ask for a given environment whether it is necessary to have the same degree of correctness assurance for both the MAC and DAC enforcement mechanisms.

While the Sea View work is presented as an example of a high assurance system utilizing a balanced assurance approach, the document focuses primarily on how one can implement DAC policies with a high assurance mechanism. For DAC controls on databases or tables the complexity is similar to operating system DAC mechanisms. As the document shows, even the complexity introduced by the WITH GRANT and CASCADE options in the GRANT/REVOKE commands can be handled with a high assurance mechanism. However, when access based on content is controlled, as is done with views, the complexity of the DAC enforcement mechanism can become too great to implement with a high assurance mechanism. The various approaches discussed in Section 5 show how at least some degree of view-based access control can be implemented with high assurance. It should be noted that for evaluation against the TCSEC, a DBMS is not required to include view interpretation in its DAC policy.

While work to date has provided a good framework for considering high assurance DAC issues and approaches to providing high assurance DAC, work on implementing useful high assurance DBMSs in the future will provide the experience needed to determine the effectiveness of these approaches and evolve new approaches. Furthermore, the use of these high assurance DBMSs in working environments will provide the insights needed to effectively tradeoff the flexibility of DAC mechanisms with the degree of correctness assurance required.

REFERENCES

[Abrams 91] Abrams, M. D., Heaney, J. E., King, O., LaPadula, L. J., Lazear, M. B., Olson, I. M., “Generalized Framework for Access Control: Towards Prototyping the ORGCON Policy,” *Proceedings of the 14th National Computer Security Conference,* Washington, DC, 1-4 October 1991.

[ANSI 92] *Database Language SQL,* American National Standard X3.135-1992, American National Standards Institute, 1992.

[Audit 96] National Computer Security Center, *Auditing Issues in Secure Database Management Systems,* NCSC Technical Report-005, Volume 4/5, May 1996.

[Codd 70] Codd, E. F., “A Relational Model of Data for Large Shared Data Banks,” *Communications of the ACM,* June, 1970, pp. 377-387.

[Date 89] Date, C. J., *A Guide to the SQL Standard,* 2nd Edition, Addison-Wesley, 1989.

[Denning 87] Denning, D. E., Lunt, T. F., Schell, R. R., Heckman, M., Shockley, W. R., “A Multilevel Relational Data Model,” *Proceedings of the IEEE Symposium on Security and Privacy,* April 1987, pp. 220-234.

[DoD 85] Department of Defense, *Department of Defense Trusted Computer System Evaluation Criteria,* DOD 5200.28-STD, December 1985.

[Entity 96] National Computer Security Center, *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems,* NCSC Technical Report- 005, Volume 2/5, May 1996.

[Fagin 78] Fagin, R., “On an Authorization Mechanism,” *ACM Transactions on Database Systems,* 3(3), 1978.

[Garvey 88] Garvey, C. and Wu, A., “ASD-Views,” *Proceedings of the 1988 IEEE Symposium on Security and Privacy,* April, 1988.

[Graubart 89] Graubart, Richard, “On the Need for a Third Form of Access Control,” *Proceedings of the 12th National Computer Security Conference,* Baltimore, MD, 10-13 October 1989.

[Griffiths 76] Griffiths, P. G., Wade, B., “An Authorization Mechanism for a Relational Database System,” *ACM Transactions on Database Systems, 1(3),* 1976.

[Harrison 76] Harrison, M. H., Ruzzo, W. L., Ullman, J. D., “Protection in Operating Systems,” *Communications of the ACM,* 19(8), 1976.

[Inference 96] National Computer Security Center, *Inference and Aggregation Issues in Secure Database Management Systems,* NCSC Technical Report-005, Volume 1/5, May 1996.

[Irvine 92] Irvine, C. E., Schell, R. R., Thompson, M. F., "Using TNI Concepts for the Near Term Use of High Assurance Database Management Systems," *Proceedings of the Fourth Rome Laboratory Multilevel Database Security Workshop*, Research Directions in Database Security W, June 1993.

[ITSEC 91] Commission of the European Communities, *Information Technology Security Evaluation Criteria, Provisional Harmonized Criteria,* Office for Official Publications of the European Communities, Luxemborg, June 1991.

[Knode 89] Knode, R. B., and Hunt R. A., "Making Databases Secure with TRUDATA Technology," *Proceedings of the Fourth Aerospace Computer Security Applications Conference,* December, 1989.

[Lunt 88] Lunt, T. F., Schell, R. R., Schockley, W. R., Heckman, M., and Warren, D., "A Near-Term Design for the SeaView Multilevel Database System," *Proceedings of the 1988 IEEE Symposium on Security and Privacy,* pp. 234-244, April 1988.

[Marks 94] Marks, D. G., Binns, L. J., Sell, P. J.and Campbell, J., "Security Considerations of Content and Context Based Access Controls," *Proceedings of the Tenth International Information Security Conference, IFIP SEC '94,* May 1994.

[McCollum 90] McCollum, C. J., Messing, J. R., and Notargiacomo, L., "Beyond the Pale of MAC and DAC - Defining New Forms of Access Control," *Proceedings of the 1990 Symposium on Security and Privacy,* Oakland, CA, 1990.

[NCSC 87] National Computer Security Center (NCSC), *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003, 30 September 1987.

[Poly 96] National Computer Security Center, *Polyinstantiation Issues in Multilevel Secure Database Management Systems,* NCSC Technical Report-005, Volume 3/5, May 1996.

[Sandhu 89] Sandhu, Ravi S., "Current Status of the Safety Problem in Access Control," *IEEE CIPHER,* pp. 37-46, Fall 1989.

[Sandhu 92] Sandhu, Ravi S., "The Typed Access Matrix Model," *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy,* Oakland, CA, 4-6 May 1992.

[Schaefer 84] Schaefer, M., Schell, R., "Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products," *Proceedings of the 1984 IEEE Symposium on Security and Privacy,* DoD Computer Security Center. May 1994, pp. 41-49.

[Schaefer 94] Schaefer, M., Smith, G., Halme, L., Landoll, D., *Assured DAC for Trusted RDBMSs Final Report*, ATR 94020, Arca Systems, Columbia, MD, September 1994. (Portions to be reprinted in IFIP 1995).

[Shockley 88] Schockley, W., Schell, R., “TCB Subsets for Incremental Evaluation,” *Proceedings of the Third Aerospace Computer Security Conference*, December, 1987, pp. 131-139.

[Sterne 94] Sterne, D. F., Benson, G. S., “Redrawing the Security Perimeter of a Trusted System,” *Proceedings of the 1994 IEEE Computer Security Foundations Workshop*, June 1994.

[TDI 88] National Computer Security Center, *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria (DRAFT)*, 1988.

[TDI 91] National Computer Security Center, *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria,* NCSC-TG021, April 1991.

[Tinto 92] Tinto, Mario, *The Design and Evaluation of INFOSEC Systems: The Computer Security Contribution to the Composition Discussion,* C Technical Report 32-92, NSA, June 1992.

[Wilson 88] Wilson, Jackson, “Views as the Security Objects in a Multilevel Secure Database Management Systems,“*Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, CA, April 1988.

[Zloof 77] Zloof, M. M. “Query by Example: a Data Base Language,” *IBM Systems Journal*,16:4, 1977, pp. 324-343.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE May 1996	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Discretionary Access Control Issues in High Assurance Secure Database Management Systems	5. FUNDING NUMBERS
6. AUTHOR(S)	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) National Security Agency Attn: V21, Partnerships and Processes Fort George G. Meade, MD 20755-6000	8. PERFORMING ORGANIZATION REPORT NUMBER NCSC Technical Report - 005 Volume 5/5 Library No. S-243,039
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report is the fifth of five companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*. The companion documents address topics that are important to the design and development of secure database management systems, and are written for database vendors, system designers, evaluators, and researchers. This report addresses discretionary access control issues in high assurance secure database management systems.

14. SUBJECT TERMS Discretionary access control; High Assurance Secure Database Management Systems	15. NUMBER OF PAGES 44
	16. PRICE CODE

17. SECURITY CLASSIFCATION OF REPORT Unclassified	18. SECURITY CLASSIIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT