Security Issues in the Database Language SQL

W. Timothy Polk Lawrence E. Bassham

July 30, 1993

Abstract

The Database Language SQL (SQL) is a standard interface for accessing and manipulating relational databases. An SQL-compliant database management system (DBMS) will include a minimum level of functionality in a variety of areas. However, many additional areas are left unspecified by the SQL standard. In addition, there are multiple versions of the SQL standard; the functionality will vary according to the particular version.

This document examines the security functionality that might be required of relational DBMS's, and compares them with the requirements and options of the SQL specifications. The comparison will show that the security functionality of an SQL-compliant DBMS may vary greatly. A variety of security policies are considered which can be supported by SQL. The document ends by showing which types of functions are required by the examined security policies.

Contents

1 Introduction **1**

1.1 Audience 1
1.2 The Standards 1
1.3 Using This Document 2
1.4 Security Considerations 3

2 SQL Architecture **5**

2.1 SQL Functionality 5
2.2 SQL Implementation 6
2.3 Security Responsibilities: The SQL Component 7
2.4 Security Responsibilities: Non-SQL Components 8
2.4.1 Application Interface to SQL 8
2.4.2 SQL Interface to Physical Database 8
2.4.3 SQL Interface to Non-SQL DBMS 9
2.4.4 Interface to Remote Databases 9

3 Security Policy **11**

3.1 Discretionary Access Control 11
3.1.1 Privileges 11
3.1.2 Authorization Identifier 13
3.1.3 Roles 14
3.2 Mandatory Access Control 14
3.2.1 Polyinstantiation 15
3.2.2 TCB Subset Architecture 17
3.2.3 Trusted Subject Architecture 18
3.2.4 Integrity Lock Architecture 18
3.3 Schema Manipulation 20
3.4 Integrity Constraints 20
3.4.1 Table Constraints 20
3.4.2 Column Constraints and Check Constraints 22
3.4.3 Assertions 23
3.4.4 Domains 23
3.4.5 The SQL'89 Security Bug 24

3.5 Object Reuse 24
3.6 Labels 24
3.7 Inference 25
3.8 Aggregation 26

4 Accountability **27**
4.1 Identification & Authentication 27
4.2 Auditing 28

5 Assurance **29**
5.1 Testing and Evaluation 29
5.1.1 FIPS Conformance 30
5.1.2 NCSC Evaluation 31
5.2 Reliability 31
5.2.1 Fault Tolerant Systems 31
5.2.2 Disk Array Technology 32
5.3 Transaction Management (Integrity) 33
5.4 Diagnostics Management 34

6 Summary/Recommendations **35**

1 Introduction

Federal agencies maintain an increasing amount of valuable and sensitive information in relational database management systems (DBMS). These agencies are required to utilize Federal Information Processing Standard (FIPS) 127-compliant database management systems. FIPS 127 specifies the Database Language SQL (SQL)[1] for accessing and manipulating relational databases.

SQL requires certain levels of functionality in schema specification, retrieval and modification of data, and transaction management. However, a number of security-relevant areas are not addressed. As a result, SQL-compliant DBMS systems offer varying levels of security functionality.

This document examines the various security aspects of SQL. Security-relevant features are identified, in conjunction with the version of the SQL standard that supports them. Critical but unspecified security features are noted, as well as the types of mechanisms that could be offered by vendors.

Finally, three broad security policies are examined. The level of support offered by the various SQL versions is contrasted for each policy, and critical controls unspecified by any version of SQL are identified.

1.1 Audience

This document is intended to assist information technology (IT) managers in the selection of DBMS's with appropriate security functionality. IT managers with knowledge of security policies and mechanisms, and familiarity with DBMS's will find it most useful. However, the document does not assume that familiarity. Background information regarding both security and databases is included to assist the reader.

1.2 The Standards

SQL is a widely used language for accessing and manipulating relational databases. Several levels of SQL are defined; these levels are generally upwardly compatible. Certain security-relevant features are required in an SQL-compliant DBMS. Other security features are not specified by SQL, but may appear in particular products. The exact functionality of an SQL-compliant DBMS varies according to the particular

[1] SQL is not an acronym, although it derives from "Structured Query Language." The complete name is *Database Language SQL*.

SQL specification and the set of unspecified enhancements which are also included.

The basic SQL definition is ANSI X3.135-1989, *Database Language - SQL with Integrity Enhancement* [ANS89a], and will be referred to as SQL'89. The functionality of SQL'89 includes schema definition, data manipulation, and transaction management. SQL'89 and ANSI X3.168-1989, *Database Language - Embedded SQL* [ANS89b], form the basis for FIPS 127-1 [FIP90].

ANSI X3.135-1992 [ANS92] describes an enhanced SQL, known as SQL'92. The enhancements include schema manipulation, dynamic creation and execution of SQL statements, and network environment features for connection and session management. FIPS 127-2 [FIP93] is based upon X3.135-1992.

Finally, a third version of SQL is currently under development in ANSI and ISO. This version will be referred to as SQL3 in the remainder of this document. SQL3 enhancements will include the ability to define, create, and manipulate user-defined data types in addition to tables.

ISO/IEC Draft International Standard 9579-1 [ISO90a] and 9579-2 [ISO90b] define the Remote Database Access (RDA) standard. RDA provides a method for interconnecting database management systems. 9579-1 describes the generic model; 9579-2 presents the SQL specialization information.

1.3 Using This Document

Section 2, *SQL Architecture*, provides an overview of the functionality and interaction of the components of a system supporting an SQL-compliant DBMS. The section includes a survey of the security responsibilities of each component in such a system.

Sections 3, 4, and 5 present required features and enhancemants of SQL-compliant DBMS systems.[2] These sections are structured to reflect the *Security Requirements* described in [TCS85]. Section 3 presents mechanisms that can be used to enforce *Security Policy*. Section 4 addresses *Accountability* mechanisms. Section 5 includes *Assurance* measures. (The TCSEC security requirement *Documentation* is omitted.)

The content of these sections does not strictly adhere to the TCSEC security requirements. Items are added or omitted to reflect the requirements of non-DoD federal agencies. Modifications include:

[2] "Enhancements" are features which are not specified in the SQL specifications, but are not ruled out by the standard either. The vendor has the option of including such features for product differentiation.

- augmenting the integrity requirements in security policy;
- omission of covert channels;
- inclusion of fault tolerant hardware, such as Redundant Array of Inexpensive Disks (RAID) storage units;
- discussion of assurance value of FIPS conformance testing; and
- discussion of inference and aggregation.

The final section of this paper presents a review of required and optional security features. These features are examined in conjunction with general security choices (e.g., mandatory vs. discretionary access control). The level of support offered by the various SQL versions is contrasted for each of these choices, and critical controls that are not specified by any version of SQL are identified.

1.4 Security Considerations

The basic security requirements are, as always, preservation of confidentiality and integrity while maintaining availability. There are a number of specific threats within these categories that merit special consideration here.

Inference and aggregation are usually considered threats to mandatory access control policies. There are also a number of DBMS specific security issues, such as referential integrity and polyinstantiation. Classic operating systems problems such as deadlock and transaction completion problems must also be considered.

The following definitions will be used in this document:

- inference: Derivation of new information from known information. The inference problem refers to the fact that the derived information may be classified at a level for which the user is not cleared. The inference problem is that of users deducing unauthorized information from the legitimate information they acquire.[Thu92]

- aggregation: The result of assembling or combining distinct units of data when handling sensitive information. Aggregation of data at one sensitivity level may result in the total data being designated at a higher sensitivity level.[Rob91]

- polyinstantiation: Polyinstantiation allows a relation to contain multiple rows with the same primary key; the multiple instances are distinguished by their security levels.[SFD92]

- referential integrity: A database has referential integrity if all foreign keys reference existing primary keys.[Cam90]

- entity integrity: A tuple in a relation cannot have a null value for any of the primary key attributes.[DJ92]
- granularity: The degree to which access to objects can be restricted. Granularity can be applied to both the actions allowable on objects, as well as to the users allowed to perform those actions on the object.

An example of polyinstantiation is included in section 3.2, *Mandatory Access Control*. Examples of inference and aggregation may be found in sections 3.4 and 3.5, *Inference Controls* and *Aggregation* respectively.

2 SQL Architecture

This section begins with a brief description of the functionality of SQL. Secondly, a model of an SQL implementation is presented. Finally, the security problems associated with each component of the model are highlighted.

2.1 SQL Functionality

SQL defines standard components and facilities for relational database management systems. The components of an SQL database are *schemas*, *tables*, and *views*. A schema describes the structure of related tables and views. Tables hold the actual data in the database; they consist of rows and columns. Each row is a set of columns; each column is a single data element. Views are derived tables, and may be composed of a subset of a table or the result of table operation (e.g., a join of different tables).

The SQL standard describes facilities to perform four specific functions:

- Schema Definition: Used to define the structure of the database, integrity constraints, and access privileges;
- Retrieval: Retrieve data from a database with a standard query interface;
- Data Manipulation: Populate and modify the contents of a database by adding, modifying or deleting rows and columns;
- Schema Manipulation[3]: Modify the structure, integrity constraints, and privileges associated with the tables and views in the database; and
- Transaction Management: The ability to define and manage SQL transactions.

Each of these components is related to certain security threats. Schema definition and manipulation relate to problems of inference and aggregation. Data retrieval tasks must conform to confidentiality policies. Data manipulation must conform to integrity policy. Transaction management contributes to maintaining the integrity of the database.

[3]Schema manipulation is introduced in SQL'92. These facilities are unspecified in SQL'89.

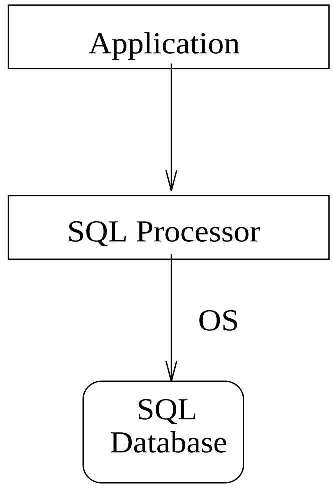

Figure 1: Standalone model.

2.2 SQL Implementation

To perform the security analysis, it is necessary to assume some architecture for an SQL implementation. Figure 1 depicts a standalone model, which can be implemented with any level of SQL. Figure 2 depicts a client/server model, which can be implemented in a standard fashion with SQL'92 or SQL3 together with RDA (ISO 9579). Client/server implementations with SQL'89 will be proprietary and may not be interoperable with other SQL products without the use of special gateways. The first model shows an application interfacing with an SQL processor, which interfaces with a physical database on a local system. This model can be implemented with any version of SQL. The second model, a simplification of the model presented in [GS92] requiring SQL'92 or SQL3, has the following features:

- An application, written in the SQL query language or utilizing embedded SQL, will communicate with an SQL server.
- The SQL server may directly access an SQL-compliant database.
- The SQL server may access a database that is not SQL-compliant through an appropriate database processor.
- The SQL server may act as a client and access a remote database.
- The SQL server may access a remote database by utilizing implementation-defined communications software and de facto standards.

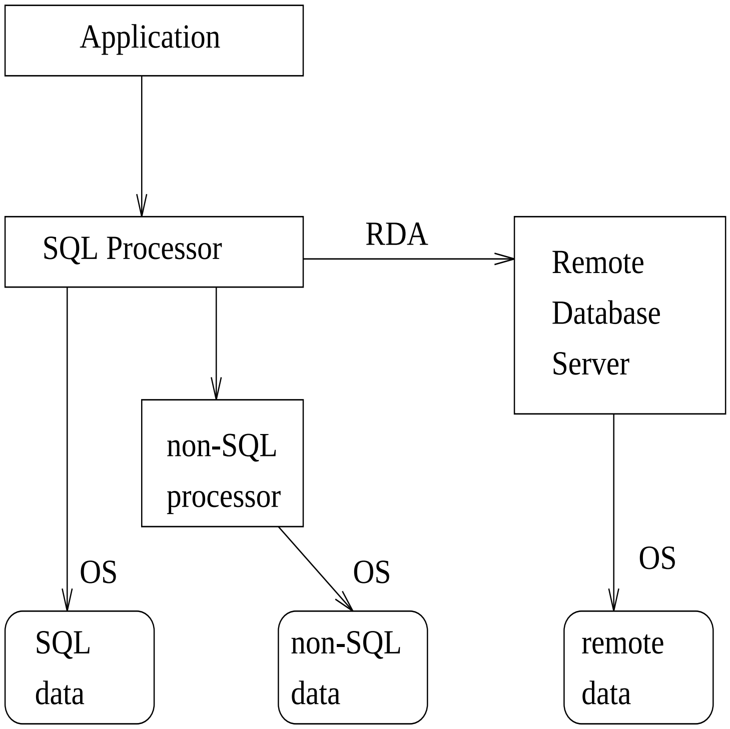

Figure 2: Client/server model.

- The SQL server utilizes the OS services of the local host to access and store data on the system.[4]

2.3 Security Responsibilities: The SQL Component

There are valid security considerations for each of the four areas of SQL functionality:

- Database Schema: The database schema must be well designed to ensure that aggregation and inference are not threats.
- Retrieval: The SQL server is responsible for maintaining access control for SQL level objects.
- Modification: The SQL server is responsible for maintaining access control for SQL level objects. The SQL server is responsible for enforcing type checking and ranges; these are external consistency issues. The SQL server is responsible

[4]In certain cases, such as database machines, the system may not have a general purpose operating system. The system will still offer certain services to the SQL layer, though.

for enforcing check constraints and uniqueness requirements; these are internal consistency issues.

- Transaction Management: The SQL server must ensure orderly access to data when concurrent transactions attempt to access and modify the same data.[5] The SQL server must provide appropriate transaction management features: incomplete transactions can result in loss of external consistency[6] - the tables and elements are no longer "synchronized."

2.4 Security Responsibilities: Non-SQL Components

2.4.1 Application Interface to SQL

The interface between the SQL processor and an application may utilize the embedded SQL language or the SQL language (interactively or invoked by a front-end processor). The application must supply accurate information regarding the identity of the user to the SQL processor. This places two requirements on the system: appropriate selection and management of identification and authentication (I&A) controls and control of this critical attribute's propagation.

If the I&A control is weak or poorly managed, there is little assurance of accuracy for this attribute. Consider passwords where the account name and password are identical (a.k.a., "joe accounts"). If an application accesses SQL for such an account, there is an increased probability that the actual user is not the authorized account user. Identity-based controls become severely weakened.

Some systems include programs or features that allow users to modify their identity. The UNIX operating system, for instance, includes the file attributes *setuid*, which re-sets the user id, and *setgid*, which re-sets the group id. Termination of the program is intended to cause the old user and group id's to resume. However, flaws in the implementation of these features may allow a user to continue to masquerade as the other user, executing SQL programs with unauthorized privileges.

2.4.2 SQL Interface to Physical Database

The operating system provides the basic services that enable the SQL processor to store, retrieve, and modify data on the system. The operating system is responsible

[5] Deadlock (and denial of service) is one possible result of such concurrent transactions; loss of data integrity is another.

[6] as defined by Clark and Wilson in [CLARK87].

for guaranteeing the simple integrity[7] of the data and preventing denial of service.

The operating system must also prevent data from being accessed outside the SQL processor. Access to raw DBMS files, database export files, or journal files may violate security policy. Such actions can result in loss of integrity (e.g., improper modification of data) or confidentiality (e.g., by circumventing internal access controls of SQL).

2.4.3 SQL Interface to Non-SQL DBMS

The SQL interface to non-SQL DBMS's is unspecified in SQL'89. SQL'92 introduces the concepts of an SQL client and an SQL server. By matching an SQL client with a non-SQL server, SQL queries may be performed on non-conforming databases; however, this requires that the non-SQL server provide an SQL-conformant view of its services and data.

The interface between a client and server must be protected. Other processes on the system could eavesdrop, insert incorrect information, or perhaps even delete information. These actions would result in loss of integrity or confidentiality.

2.4.4 Interface to Remote Databases

The RDA standard is designed as a generic interface between local and remote database servers. RDA also has an SQL Specialization for connecting SQL-compliant databases. Currently, RDA is only defined for use in an OSI network environment. Partially conformant products which use TCP/IP are also available.

Use of RDA on an open network may expose the system to many threats. Eavesdropping, packet replay, and host spoofing are likely threats. These threats can be minimized by employing encryption techniques and strong authentication measures.

The RDA standard allows for exchange of authentication data. However, it is not required. Encryption techniques may be employed at several different layers of the OSI stack. RDA does not require or forbid such techniques; therefore, the security achieved will be dependent upon the implementation.

Where proprietary protocols are used, the system will be exposed to all the same threats as use of RDA. From a security standpoint, the same threats must be addressed. (From an open systems standpoint, there are also interoperability problems.) In addition, if a translator gateway is required this may add a single point of failure to a distributed database architecture.

[7] Simple integrity refers to the operating system reading and writing data in a predictable manner.

3 Security Policy

This section addresses mechanisms for enforcing security policy on computer systems. It concentrates on controlling the access to, and modification of, data. This corresponds roughly to confidentiality and integrity policy although availability can sometimes be affected.

The section begins by addressing access control mechanisms. Discretionary and mandatory access controls are examined in turn. Next, the section reviews integrity constraints. This is followed by the more traditional security features of object reuse and labeling. The section closes by examining mechanisms for controlling inference and aggregation.

3.1 Discretionary Access Control

Discretionary access control (DAC) is a means by which access to objects is restricted to specific users or groups of users. The access control is discretionary in that access privileges may be passed on to other users, either directly or indirectly, by the owner of the object.

3.1.1 Privileges

Privileges are the means by which SQL enforces DAC. Privileges are granted with a Grant statement and are used to specify an allowable action on a specific object, e.g., to UPDATE the rows in a specific table, to a grantee.

<u>SQL'89</u>

SQL'89 defines the following five privileges which establish the granularity of access available to users of the database: *INSERT*, *DELETE*, *SELECT*, *UPDATE*, and *REFERENCES*.

- The *INSERT* privilege grants a user the ability to create new rows in a base table or a viewed table. If the new row is to be added to a base table, the candidate row must include every column of the base table for which no default value has been either implicitly or explicitly defined. If the candidate row is to be added to a viewed table, the candidate row must include every column in the base table from which the viewed table is derived. Additionally, the view must be updatable.

- The *DELETE* privilege grants a user the ability to delete rows from a table. In order to delete rows from a table or view, the delete privilege needs to have been granted and the view must not be read-only.
- The *SELECT* privilege grants a user the ability to retrieve values from a table. Essentially, the select privilege allows users to read tables.
- The *UPDATE* privilege grants users the ability to update, or change, the contents of a row in a table. In addition, to perform updates on a view, the view must not be read-only. Column constraints can be specified when granting this privilege; that is, a user may be allowed to update certain columns within a table or view.
- The *REFERENCES* privilege grants a user the ability to specify a foreign key reference across schemas. Foreign keys are fields which coincide with a unique field (e.g., primary key fields) in the grantor's table. A security implication of the references privilege is that it can be used, possibly inadvertently, to implement a denial of service attack. When table B references table A, records from table A cannot be deleted or the primary key field cannot be changed if a record from table B corresponds to that record from table A. Another security implication of the references privilege is that a user could, based on the constraint definition, determine all legal values for the referenced column. A policy violation would occur if this information was used to infer knowledge from which a user had been exluded.

Privileges can be granted to individuals or to everyone (if PUBLIC is specified). Caution should be used when utilizing the PUBLIC specifier. Additionally, privileges can be granted with the WITH GRANT OPTION. This gives the grantee of a privilege the ability to subsequently grant that privilege to other users. As a result, the owner may loose control over his own table. Finally, privileges can be granted one by one, as a comma separated list, or with the ALL specifier. The ALL specifier, however, refers only to those privileges grantable by that user.

A significant security flaw in SQL'89 is the fact that there is no standard way to revoke privileges. As job requirements change, necessary access to the database could also change. It should be noted that the standard does not exclude vendors from incorporating a statement for revoking privileges into an implementation, and many vendors do include such a statement.[8] The standard simply does not require such a statement, or specify the semantics of such a statement if included.

[8] In fact, the authors do not know of any product which omits a mechanism for revoking privilege.

SQL'92

A new privilege has been defined for SQL'92. The *USAGE* privilege is used to allow or restrict access to domains, collations, character sets, and translations. Additionally, all privileges from SQL'89 hold with the following extensions:

- The *INSERT* privilege can be specified with a column constraint as well as on whole tables. (SQL'89 specified column constraints only for the UPDATE and REFERENCES privileges.)
- The *REFERENCES* privilege has several extensions. These extensions maintain referential integrity on delete and update operations in the base table. Instead of a delete or update operation in the base table being blocked because the record is referenced in another table, the reference can be specified with one of the following actions:
  - The *CASCADE* specifier will propagate the change from the base table to the referencing table. On update, the update will appear in the referencing table. On delete, matching rows in the referencing table will also be deleted.
  - The *SET NULL* specifier will set, for both update and delete, the referencing column in all matching rows to the null value.
  - The *SET DEFAULT* specifier will set, for both update and delete, the referencing column in all matching rows to the default value, as specified with the "<default clause>."
  - The *NO ACTION* specifier performs no referential integrity function. It is included for backward compatibility reasons. It results in the same functionality as SQL'89. When no referential integrity constraint is specified, the NO ACTION specifier is implicit.

SQL'92 adds the *REVOKE* statement. With this statement, all grantable access privileges can be revoked. The revoke statement can also be used to revoke the grant option from a user. As a result that user could no longer grant privileges to other users.

3.1.2 Authorization Identifier

For both SQL'89 and SQL'92 <authorization identifiers>, the SQL non-terminal used to specify users (e.g., <grantor> and <grantee>), are defined in an implementation-dependent way. This means SQL does not define how operating system users are mapped to SQL users. See Section 4.1, *Identification and Authentication*, for more information.

3.1.3 Roles

With current versions of the SQL standard, maintenance of appropriate access control restrictions is difficult. The Grant and Revoke statements are available to allocate individual privileges, but this leaves much for an administrator to maintain. A change in job requirements by one user can require many changes to database access for that user. To complicate matters further, if the user whose job requirements changed has granted privileges to other users, those privileges must be examined for correctness.

Another drawback of the current privilege system is that a user accumulates privileges required for different job functions. For example, separate job functions for one individual may include payroll clerk and purchasing agent. It may be desirable to allow the user access to only payroll related objects while performing payroll clerk functions and purchasing related objects while performing purchasing agent functions.

SQL3, the newest version of SQL being developed, has an enhanced facility for the manipulation of access rights. In addition to the existing Grant and Revoke statements, a new construct, called Roles, is being developed. This facility will allow database administrators to create individual roles with corresponding database access requirements. Then, for example, when a user's job requirements change to no longer include payroll clerk activities, only one Role needs to be revoked instead of revoking access privileges to all objects needed only for payroll clerk activities.

An additional benefit of Roles is that a user can have only one Role active at a time. This would allow a user to access only payroll related objects when working under the Payroll Role, and access procurement related objects when working under the Purchasing Role.

3.2 Mandatory Access Control

Mandatory access control (MAC) is not supported directly in SQL. However, there are several different methods for implementing a mandatory access control model. The major architectures for trusted DBMS products [Cam90] are the Trusted Computing Base (TCB) Subset Architecture (Figure 6), Trusted Subject Architecture (Figure 7), and Integrity Lock Architecture (Figure 8). Each is intended for use with a Trusted Operating System (OS), but requires different controls.

It should be noted that an SQL-based DBMS with mandatory access controls can be designed without modification of the SQL syntax. However, certain modifications in SQL semantics must be made if polyinstantiation is used to control inference.

3.2.1 Polyinstantiation

Polyinstantiation is frequently used with mandatory access control database systems to control inference. This section is intended to explain polyinstantiation. Inference, and the application of polyinstantiation for inference control, are described in Section 3.4, *Inference*.

In the following example, the database is a single relational table. The table contains two columns: Patient name and Disease. The Patient name field is the key for this table. There are two clearance levels, HIGH and LOW. Two sets of data exist; the first set is HIGH data (Figure 3) and the second is the LOW data set (Figure 4).

Patient Name	Disease
Howard	AIDS
Gordon	Syphilis
Jackson	Gunshot wounds

Figure 3: HIGH data.

The HIGH data include patients under police guard, such as Jackson, or patients with confidential diseases. The LOW data include all other patients, and perhaps some of the HIGH patients with different data.

Patient Name	Disease
Smith	Lung Cancer
Howard	Pneumonia
Jones	2nd degree burns
Hamp	heart failure

Figure 4: LOW data.

When users with LOW security level browse the database, they are only permitted to see the LOW data. If a user wishes to add a LOW record with primary key X, the command is accepted even if a HIGH record exists with that key.

When a user with HIGH security level browses the database, he sees all of the HIGH records, as well as the LOW records with a primary key that is not found in the HIGH

Patient Name	Disease
Howard	AIDS
Smith	Lung Cancer
Gordon	Syphilis
Jackson	Gun Shot wound
Jones	2nd degree burns
Hamp	heart failure

Figure 5: HIGH user's view.

data. The resulting table is shown in Figure 5. Note that the record for Howard does not appear twice; only the HIGH level record appears.

This feature may be useful in a number of ways. LOW users cannot determine if a HIGH record exists with key Gordon by attempting to create a record and checking for an error message. Dual records could be used, as in the case of Howard, to prevent LOW users from discovering the true nature of Howard's illness. This is intended to prevent disclosure by inference.

In many situations, polyinstantiation may be implemented by a local database security administrator using only standard features from SQL'89. The above example is easily implemented with two base tables, known only to the security administrator, and a single view available to all other users.

```
BaseTable1(PatientName,Disease,Level) with Primary Key(PatientName,Level)
BaseTable2(UserName,SecurityLevel) with Primary Key(UserName)



CREATE VIEW PatientInfo(PatientName,Disease)
     AS SELECT PatientName,Disease
     FROM BaseTable1
     WHERE BaseTable1.Level = (SELECT SecurityLevel FROM BaseTable2
                         WHERE UserName = CURRENT_USER
                    )
          OR
          (BaseTable1.Level = "LOW"
               AND
               NOT EXISTS (SELECT * FROM BaseTable1 AS X
                    WHERE X.PatientName = BaseTable1.PatientName
```

```
                AND X.Level = "HIGH"
            )
        )
```

With view optimization techniques available in most SQL-conformant processors, user access through the PatientInfo view should suffer no significant performance penalty over direct access to BaseTable1.

This view is always updateable, provided that a DEFAULT value exists for Level in BaseTable1. In SQL'89 and SQL'92, the default clause does not allow a case expansion to determine the default, so one cannot specify in the schema that insert values for Level are HIGH for high level users and LOW for low level users. Instead, one would specify a default and all new inserts would have that Level initially assigned.[9]

Ensuring that new inserts are all assigned the appropriate security level requires a second view, to be used only by HIGH level users. The second view would assign inserts a HIGH value for Level by default. The first view would have a default of LOW for Level.

3.2.2 TCB Subset Architecture

The TCB Subset model implements MAC by maintaining the database in multiple, single-level, files. The operating system enforces the access control policy, restricting the DBMS process to appropriate information. This means that the DBMS does not have to be trusted, so evaluation is simplified. The DBMS might still enforce privilege based access controls, but would not enforce MAC policy. If no DAC policy is required, all users would have all privileges for all tables.

The TCB Subset model implies polyinstantiation. If a record r_1 exists with keys K, but is classified SYSTEM HIGH, a SYSTEM LOW user cannot see it. If a SYSTEM LOW user attempts to add a record r_2 with keys K, the system must do so. Now the DBMS has two records with the same set of keys in the same table. SYSTEM LOW users will see r_2. SYSTEM HIGH users will see r_1 instead; r_2 is considered incorrect. This can reduce inference problems, but results in a variety of integrity problems. The data in the two records may become out-of-date. Then if one record is changed, but the other is not, the DBMS loses integrity.

Note that this is quite complicated if categories are being utilized. The TCSEC suggests that MAC systems support a minimum of eight security levels and 256

[9]The emerging SQL3 specification includes facilities that easily get around this problem.

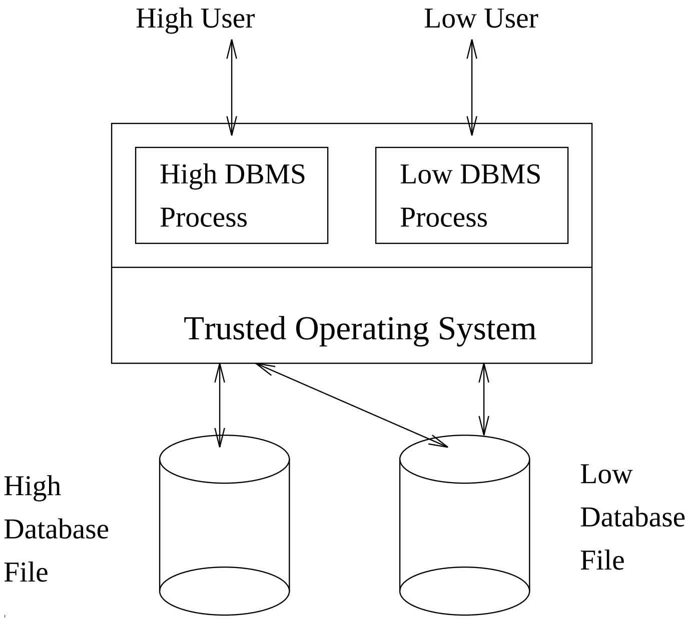

Figure 6: TCB Subset Architecture.

categories. Since the files must support combinations of categories for each level, the number of files is unmanageable. To avoid this problem, the DBMS could have to enforce the category aspect, but this defeats the purpose of the architecture: the DBMS process must be a trusted process.

3.2.3 Trusted Subject Architecture

A Trusted Subject Architecture DBMS enforces both MAC and DAC. The database is stored on the system as SYSTEM HIGH OS objects. Within those files, DBMS objects are labeled according to security policy. Those labels are used as the basis for MAC enforcement. DAC enforcement is based upon the usual SQL DAC specifications.

The Trusted Subject Architecture does not imply or rule out polyinstantiation. Support for polyinstantiation must be built in if it is required.

3.2.4 Integrity Lock Architecture

The Integrity Lock Architecture uses an untrusted DBMS in conjunction with a trusted OS and trusted filter to enforce security policy. The DBMS could enforce DAC policy, but MAC policy and labeling would be enforced by the trusted filter. En-

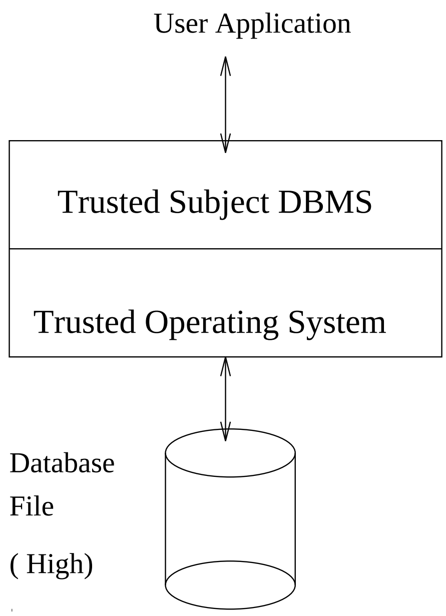

Figure 7: Trusted Subject Architecture.

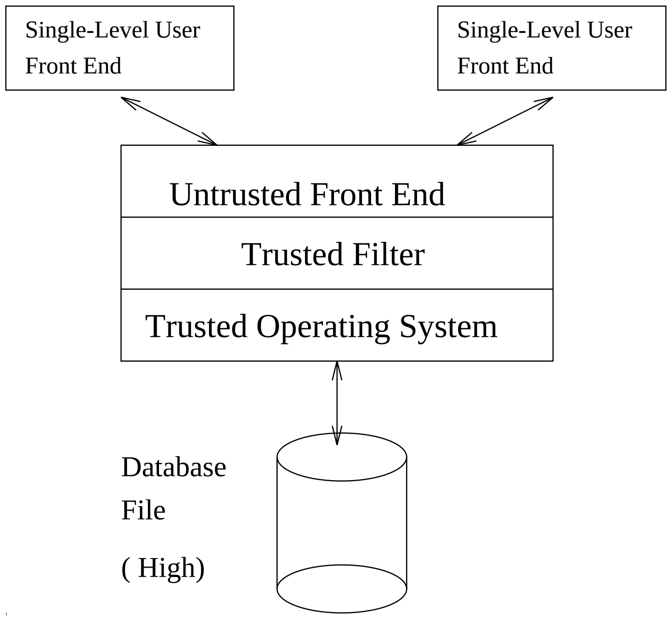

Figure 8: Integrity Lock Architecture.

cryption and cryptographic checksums are employed to protect the security label from modification. The Integrity Lock Architecture implies support for polyinstantiation. This architecture allows use of off-the-shelf DBMS software. It has disadvantages due to high overhead.

These are the most common architectures for MAC DBMS's, but are not the only ones. They can also be combined to some extent. For instance, the Integrity Lock Architecture could be used to add category enforcement to the TCB Subset Architecture.

3.3 Schema Manipulation

SQL'92 allows manipulation of the schema itself, rather than just the data. Columns and constraints on columns can be added or removed from tables. Additionally, schema object, such as domains and constraints, can be altered or deleted. Also new to SQL'92 are schema definition tables. These tables are created by the SQL processor and are treated as views, in that they can be accessed but not directly changed in SQL.

3.4 Integrity Constraints

Data integrity is addressed by a variety of data constraints specified in the database schema. These constraints describe relationships between tables, relationships between rows in a table, and permissible values for elements. Relationships between tables, or between rows in a table, are known as *table constraints.* Range checks and other specifications for data values are *element constraints.*

3.4.1 Table Constraints

SQL'89

SQL'89 defines three types of integrity constraints that may be placed upon tables. These constraints are:

- unique constraints;
- referential constraints; and
- check constraints.

T-1				
T_i	a	b	c	d
T_1	2	2	3	4
T_2	2	2	4	4
T_3	2	2	3	5
T_4	6	2	3	4

T-2				
T_i	a	b	c	d
T_1	2	2	3	4
T_2	2	2	4	4
T_3	2	2	3	5
T_4	2	6	3	4

Table 1: Uniqueness examples: contents of T-1 and T-2

SQL'89 also defines *with CHECK option* on views.

A *unique constraint definition* specifies a list of columns in the table T. T cannot contain multiple rows where the values of each of the corresponding columns are identical. Each of the listed columns must be defined as NOT NULL.

Example: Assume T has columns {a,b,c,d} and is constrained for uniqueness on {a,c,d}. Row T_i has values $\{a_i, b_i, c_i, d_i\}$. T meets the constraint if there are no rows T_j and T_k such that $\{a_j = a_k,\ c_j = c_k,$ and $d_j = d_k\}$. In Table 1, T-1 meets the constraints, but T-2 does not. Rows T_1 and T_4 are not unique for the specified columns.

A *referential constraint definition* specifies columns in *T* which reference keys in another table *F*. If all specified columns in a row of T are non-null, then a row in F must exist such that all corresponding columns match. The table has referential integrity if every row meets this criteria, or has a null value in the specified columns.[10]

A *check constraint definition* specifies a condition which all rows in T must satisfy. The condition may restrict a column, or may restrict relationships between columns. (For example, within a row MAX-TEMP >= MIN-TEMP.) The condition is ternary; it may evaluate to "true," "false," or "unknown." The condition may specify illegal values or legal value ranges. The table does not satisfy the check condition if and only if there exists a row for which the condition evaluates to "false."

Views with check options are similar to check constraints upon tables. However, the constraint is satisfied if and only if the condition evaluates to "true."

SQL'92

SQL'92 enhanced referential constraints with the MATCH FULL and MATCH PAR-

[10]If one or more specified columns in the row are null, correspondence can not be established.

TIAL specifications. If no match type is specified, the functionality is identical to SQL'89. If MATCH FULL is specified, then for each row in T:

- all referencing columns must be null; OR
- all referencing columns must be non-null and there must be a row in F such that all corresponding referencing columns are equal value.

If MATCH PARTIAL is specified, then for each row in T:

- there must be a row in F such that all corresponding referencing columns are equal value, or a referencing column value in T is null.

3.4.2 Column Constraints and Check Constraints

SQL'89 defines six types of integrity constraints that may be placed upon columns. These constraints are:

- data type;
- precision;
- references specification;
- default clause;
- CHECK constraint definition; and
- NOT NULL.

The defined data types for SQL'89 are character strings and numbers. The character strings are specified with a fixed length. Numbers may be *exact numeric value*s or *approximate numeric value*s.

Exact numeric values include integers and real numbers with a precision and scale. A real number of exact numeric value is a string of decimal digits of length *precision*. The exact numeric value is the integer value of the significant digits multiplied by 10^{-scale}.

Approximate real numbers have an exponent and mantissa. The mantissa is a signed numeric value; the exponent is a signed integer that specifies the magnitude of the mantissa. Approximate real numbers have precision; precision is a positive integer specifying the number of binary digits in the mantissa. Integers come in two sizes. Real numbers can be defined in five different ways, allowing the database designer to create fields tailored for the data.

The references specification allows the specification of a referential integrity clause for a single column (rather than a list of columns). Any non-null value entered into that column must exist in the corresponding table and column.

Default clauses may be stated explicitly or implicitly. SQL'89 permits default values of NULL, USER, or a literal (constant of appropriate data type). Additional default options, such as <datetime value function>, are available in SQL'92.

A CHECK constraint definition specifies a condition which the column element must satisfy in each row. This differs from the table check constraint; the condition can only involve the specified column element. Again, the condition is ternary; it may evaluate to "true," "false," or "unknown." The condition may specify illegal values or legal value ranges. The table does not satisfy the check condition if and only if there exists a row for which the condition evaluates to "false."

The NOT NULL constraint is an implicit check constraint. It corresponds to CHECK <column name> IS NOT NULL.

SQL'92 adds two new data types, datetime and interval. The type datetime includes DATE, TIME, and TIMESTAMP. The type interval allows specification of a year-month or day-time interval. Variable length character strings are also added in SQL'92, and the character set for the character string may also be defined.

User-defined data types are under consideration for SQL3.

3.4.3 Assertions

In SQL'92, *assertions* enhance the SQL'89 check constraints. Assertions are named constraints that "may relate to the content of individual rows of a table, the entire contents of a table, or to a state required to exist between a number of tables." [FIP93] This is a significant enhancement, since SQL'89 check constraints applied to column(s) in a single row. (In SQL'92, check constraints are allowed to contain subqueries.)

3.4.4 Domains

Domains were introduced in SQL'92. Domains are a significant enhancement for data abstraction used to specify a set of permissible values. Domain definitions can also be used to enforce a variety of format constraints, such as position of hyphens in a date field.

3.4.5 The SQL'89 Security Bug

SQL'89 allowed a user with UPDATE or DELETE privileges to use WHERE clauses even if they did not have SELECT privileges. This allowed users to "probe" the system for data they were not privileged to have. They could confirm the existence of a record with certain column values even though they could not directly read the record. This bug was fixed in an SQL'89 Errata and is specified correctly in SQL'92.

Note that a workaround exists for older SQL'89 systems. By defining updatable views, the columns users can access may be limited to the appropriate subset of the data. In any case, a user with DELETE privileges will be able to determine values of primary keys.

3.5 Object Reuse

Object reuse is defined in [Rob91] as:

> The reassignment to some subject of a medium (e.g., page frame, disk sector, magnetic tape) that contained one or more objects. To be securely reassigned, such media must contain no residual data from the previously contained object(s).

SQL'89 and SQL'92 have no specification regarding object reuse. In order to accommodate object reuse issues, both the SQL processor and the operating system will need to address the issue jointly. The operating system is responsible for deallocating system resources, such as files used to store whole tables, and the SQL processor is responsible for deallocating SQL objects, such as individual rows of tables. To maintain confidentiality, data stored in these resources and objects must be zeroed out or replaced with random data before reassignment.

3.6 Labels

Mandatory access controls require support for security labels. These labels are used as the basis for access control decisions. In order to correctly label data, the system must request and receive the security level of data. This can be accomplished in several ways.

On a trusted system, the user may specify the security level of each session. (The specified security level must be "less than" or equivalent to the user's clearance level, of course.) That information would be passed to the DBMS, and all input would be labeled at that level by default. Alternatively, the DBMS could include a mechanism to request and set the current level interactively.

If the DBMS is not trusted, as in the Integrity Lock Architecture, all mechanisms regarding labeling will be placed in the trusted filter or operating system.

In addition, if the DBMS labels DBMS objects with greater granularity than OS objects, the DBMS must maintain label integrity. For instance, if all DBMS objects are stored in a single SYSTEM HIGH OS object, as in Figure 7, then the DBMS must maintain labels for the DBMS objects. This is in contrast to Figure 6, where the OS keeps track of all labeling information.

3.7 Inference

Consider a research hospital, which has a database of doctors and patients. Patient information includes address, Social Security number, doctor name, known allergies, current prescriptions, and scheduled appointments. Each patient's medical history is kept on-line in a series of medical records. Scheduled hours, appointments, and specialty are associated with each doctor.

This database is used for scheduling appointments, billing patients, and filling prescriptions. The hospital wishes to protect the patients' anonymity, and prevent disclosure of their ailments to anyone other than a patient's individual doctor. For this reason, the average user is not allowed to access the patient history/medical records.

However, the database may still be vulnerable to disclosure through *inference*.

- Doctors generally specialize in the treatment of particular diseases. It may be possible for hospital staff to *infer* a patient's ailment from the identity of the doctor. This could be determined by viewing the patient information *or* the doctor's schedule.

- Drugs are generally associated with the treatment of particular diseases. It may be possible for hospital staff to *infer* a patient's ailment from the prescription.

Polyinstantiation is the key method used in these situations for limiting inference in multi-level secure systems.

3.8 Aggregation

Consider a database of parts for a missile. Each part's information includes sufficient information for a manufacturer to fabricate the part. This information would include materials, physical geometry, and finishing treatment(s). (A screw might be described as follows: steel; 1 x 8 pan head with fine left-hand threads; rust-inhibiting paint). In addition, the database includes assembly information and the quantity of each part required to assemble one missile.

Each part is unclassified. The combined schematic and missile design are classified as SECRET. If each manufacturer is limited to accessing a few part descriptions, he will not learn anything about the missile itself. However, if the manufacturer can access the entire database, they may be able to figure out how to build the missile.[11]

To limit aggregation, one should limit access as tightly as possible. Inference is a problem that is derived primarily from poor database design. There are several methods for detecting and reducing the potential for disclosure by inference, including those described in [Thu92]. These methods can be used in conjunction with SQL, but could not be performed within the confines of SQL itself. These methods require additional information regarding the relationships between the elements of different relations.

The inference problem gets the most attention in MAC environments, but can occur in DAC systems as well. Fortunately, the same tools should apply to DAC systems.

The aggregation problem occurs when two pieces of information A and B are classified at level X individually, but level Y (Y higher than X) collectively. This problem may in fact be insolvable. Denying access to A if User 1 with clearance X has already viewed B would require an infinite history, and quickly leads to inference problems. Aggregation is primarily a MAC problem.

[11] It is a lot like assembling a bicycle on Christmas Eve without directions. You know what a bicycle looks like and you have a pile of parts. You just try to use them all.

4 Accountability

Accountability is not addressed in the SQL specifications. Accountability is primarily achieved with two classes of mechanisms: identification and authentication controls; and auditing.

4.1 Identification & Authentication

Identification and authentication (I&A) mechanisms are not specified in SQL. However, they are required implicitly. The DAC mechanisms all assume that such information is available. Such mechanisms are usually provided by the host system, and the information is passed to the SQL processor. (In SQL'89 this is implementation defined; in SQL'92 it is performed via the CONNECT statement.) The quality of this information will vary according to authentication technique and when that authentication is performed.

In the simplest case, the user authenticates his identity to the system at logon. That information is maintained throughout the session. The information is passed to the DBMS when the DBMS is accessed. The strength of authentication varies with the type, implementation, and management of the authentication mechanisms.

That information may be incorrect by the time the DBMS is accessed. The user may have left the terminal or workstation unattended, and another person may actually be at the keyboard. The information may be improved if the user re-authenticates when the DBMS session begins.

A stronger method requires re-authentication with every transaction. This is too burdensome for systems relying on passwords, but smart card based systems can support this requirement. This method provides high assurance that the user identification was correct at the time the transaction was initiated.

If such mechanisms are not supplied by the host system, the SQL processor could incorporate its own I&A mechanism. However, the lack of I&A implies a lack of access control by the host. In such a case, the processor would have to utilize encryption to protect the data. That is, a host without access control requires a DBMS based upon the Integrity Lock Architecture.

Selection of I&A mechanisms must be tempered with common sense, of course. If passwords will be used to authenticate both system and DBMS session, the same mechanism (and password) should be used. Requiring users to remember multiple passwords is likely to result in misuse (e.g., they will write them down). Both passwords and biometrics are inappropriate for authenticating transactions; the burden upon the user is too great.

4.2 Auditing

Auditing concerns for trusted database systems are described in [SFD92] as follows:

> Auditing of security-related activities is required in [trusted] DBMS's. Security-relevant events include logins, granting and revoking of access permissions to relations, etc. The level at which auditing needs to be done is variable. The performance effects of the optional auditing features should be carefully examined, since their use may be a significant factor in the performance of data management functions.

This statement is equally applicable to all database systems.

The SQL specification does not include auditing requirements, but SQL products may include some auditing functionality. If the SQL processor includes auditing, the OS must have sufficient access controls to prevent modification of, or access to, the audit trail.

Warning mechanisms are closely related to auditing requirements. Such mechanisms notify the system or DBMS administrator if critical events occur. (An example might be an attempt to access tables without sufficient privilege.) Again, SQL has no requirement for such mechanisms, but processors may include them.

5 Assurance

Assurance describes a broad range of mechanisms and procedures. These mechanisms and procedures address the behavior of the system. A system with high assurance is more likely to operate as expected than a system with low assurance. Unexpected behavior is generally the result of hardware failure or software bugs. Hardware failure may be subtle (such as data transmission errors) or catastrophic (such as a disk crash). Software bugs may be in the OS, DBMS, or locally developed applications.

This section examines four areas of assurance:

- testing and evaluation;
- reliability of hardware;
- SQL transaction management; and
- SQL diagnostic reporting.

5.1 Testing and Evaluation

Two primary sources for assurance that software functions as expected are testing and formal evaluation of software. Testing is performed by supplying inputs to the DBMS while it is in various states, and analyzing the results. Formal evaluation involves review of design specifications and code as well as testing.

Testing can be very informal, consisting of execution of a few test applications, or quite rigorous. The features that are specified in the versions of the SQL standard can be tested in a very structured fashion to demonstrate compliance to the standard. This is known as *conformance testing*. FIPS conformance testing is a primary source of this information. NIST maintains the NIST SQL Test Suite for validating conformance to FIPS SQL.

Formal evaluation involves reviewing the architecture, source code, and documentation to detect any flaws in the system. This process may include formal verification of design and program correctness. Formal evaluation by third parties currently concentrates upon security functionality in general, and confidentiality in particular, due to the expense. However, the process could be performed against any standard or standard set of criteria.

The primary source for formal evaluations by third parties has been the National Computer Security Center (NCSC). The NCSC performs evaluations of systems and subsystems against the *Department of Defense Trusted Computer System Evaluation Criteria* (TCSEC), which is commonly known as the Orange Book. Other organizations providing formal evaluations are the European Community evaluations against the *Information Technology Security Evaluation Criteria* (ITSEC) and the Canadian Security Establishment evaluations against the TCSEC and Canadian security criteria. In each case, the evaluations are limited to security-relevant features (as defined by the particular criteria).

Conformance testing and security evaluation of SQL processors are complimentary processes. For example, NCSC evaluations do not look at code that falls outside the "trusted computing base" of security-relevant code. As a result, NCSC evaluation will not review many areas of SQL functionality that are reviewed in FIPS conformance testing. The TCSEC supplies criteria in many areas where the SQL standard is silent, so NCSC evaluations will cover many areas that are outside the scope of conformance testing.

5.1.1 FIPS Conformance

FIPS 127-1 specifies ANSI X3.135-1989, *Database Language - SQL with Integrity Enhancement*, and X3.168-1989, *Database Language - Embedded SQL*.[12] FIPS conformance testing provides assurance that the features specified in the SQL standard function as expected. This testing addresses the entire range of SQL functionality, not just security.

Conformance testing is performed by executing a suite of test programs and evaluating the results.[13] Conformance testing does not review the code itself. Security flaws such as trapdoors or bugs in the code may not be detected. The evaluation is platform independent, so platform dependent security problems will probably go undetected. Many security concerns, such as covert channels, are simply not an issue in this type of testing.

Conformance testing is also "flat" with respect to MAC. The database is defined, populated, and queried at a single security level. This makes sense, since SQL does not define any functionality regarding MAC. However, polyinstantiation would violate SQL's uniqueness clauses, but is not detected because of the single level testing method.

SQL'92 has been approved as both ANSI and ISO standards. FIPS 127-2, which adopts ANSI X3.135-1992, was approved in June 1993. Conformance tests have been developed only for Entry Level SQL'92, because it represents a minor enhancement

[12] FIPS 127-1 does not require all features of SQL'89: see the standard itself for details.

[13] The conformance testing procedures are described in [GS92].

over the SQL'89 tests. Conformance testing for Intermediate and Full SQL'92 represent a major development effort which has not yet begun.

5.1.2 NCSC Evaluation

The TCSEC and [TDI91] provide the basis for NCSC security evaluations of trusted database management systems. These evaluations consider security policy[14], accountability, audit, and assurance. NCSC evaluations are platform dependent, and result in certification of a DBMS for use on a particular computer system, with a particular operating system. A number of issues, such as covert channels, are addressed where results are invalidated by changes in the platform.

Evaluations may consider a wide range of assurance levels. At higher levels, NCSC evaluations include review of security relevant code (the TCB). At the highest levels, NCSC evaluations require formal verification of all code in the TCB.

5.2 Reliability

The most valuable asset in a database system is often the data itself. Loss of access to this data may be measured per minute in some cases. This loss, known as *denial of service*, may arise from a number of situations. One of these is the physical failure of hardware.

Insuring reliability of hardware is the primary technique to address hardware failure. Fault tolerant systems address system failure; disk array technology can be used to address storage media failure. Fault tolerance is not required by any SQL specification, but is a feature of certain SQL implementations.

5.2.1 Fault Tolerant Systems

If the hardware platform itself is down, there will be no access to the system. Fault tolerant systems are designed to continue correct operation in the event of failure of any single component. They typically exhibit both redundancy physically and conceptually (two or more CPU's, buses, disk controllers, etc.) and perform fault-detection tests with error-detecting codes, disagreement detectors, and self-checking logic circuits. They rely on disk array technology and WAFER storage technologies[15]

[14]The TCSEC security policy criteria is weak in the area of integrity policy.

[15]WAFER storage are solid state storage subsystems. They are expensive, but have performance advantages when compared to disk array peripherals.

for fault tolerance in their peripherals.

Many systems are not designed to be completely fault tolerant, but include redundancy at common points of failure. These systems might be called fault resistant. For example, a disk cabinet with four disk drives might include redundant power supplies. A single power supply would be sufficient to run the disks; however, a power supply failure would take them all out of service. Redundant supplies mean two power supply failures are required to halt the systems.

This feature does *not* make the disk drives in the cabinet fault tolerant; a single drive failure will result in loss of that disk's data and storage. It does make failure due to power supply less likely. Fault tolerance for disks requires use of disk array technology.

5.2.2 Disk Array Technology

Disk array technology uses several disks in a single logical subsystem. Disk arrays were introduced in [KGP89], which described six classifications for "Redundant Array of Inexpensive Disks," or RAID systems. They were numbered RAID-0 through RAID-5. RAID-1 through RAID-5 offer varying degrees of security functionality.[16]

Disk Shadowing

To reduce or eliminate downtime from disk failure, DBMS servers may employ *disk shadowing* or *data mirroring*. A disk shadowing, or RAID-1, subsystem includes two physical disks. User data is written to both disks at once. In this case, if one disk fails, all of the data is immediately available from the other disk. Disk shadowing incurs some performance overhead (during write operations) and increases the cost of the disk subsystem since two disks are required. However, the major problem with RAID-1 is the 50% disk overhead; for every 100 megabytes of disk space, 200 megabytes are required.

RAID-2 through RAID-4

RAID levels 2 through 4 are more complicated than RAID-1. Each involves storage of data and error correction code (ECC) information, rather than a shadow copy. Since the error correction data requires less space than the data, the subsystems have lower disk overhead. Each level has its own performance implications.

[16]RAID-0 provides only balanced performance. RAID-6 and RAID-7 have also been proposed since that time.

RAID-5

RAID level 5 involves storage of data and error correction information but does not require a dedicated shadow or ECC disk. RAID-5 has good performance characteristics, since it has the ability to read and write in parallel. Unfortunately, RAID-5 implementations work best with large numbers of physical disks (10 to 20+) which rules out small to mid-sized disk subsystems.

5.3 Transaction Management (Integrity)

A database may be in a *consistent* or *inconsistent* state. A consistent state implies that all tables (or rows) reflect some real-world change. An inconsistent state implies that some tables (or rows) have been updated but others still reflect the old world.

Transaction management mechanisms are applied to ensure that a database remains in a consistent state at all times. These mechanisms allow the database to return to the previous consistent state if an error occurs. Statements available in SQL'89 for transaction management include the *rollback* and *commit* statements. These statements are used to terminate transactions. SQL3 adds the concept of *savepoints*.

- The *rollback* statement terminates a transaction and cancels all changes to the database, including data or schema changes. This returns the database to the previous consistent state.
- The *commit* statement terminates a transaction and commits all changes to the database, including both data or schema changes. This makes the changes available to other applications. If a *commit* statement cannot complete successfully, for example a constraint is not met, an exception is raised and an implicit rollback is performed. Note that both statement rejects and transaction rollbacks are permitted by the SQL standard.
- The *savepoint* feature allows a user to mark points in a transaction, creating *subtransactions*. With this feature, a user can rollback portions of a transaction without affecting other subtransactions. For examples of savepoints and subtransaction management see [Gal91, pages 23–24].

5.4 Diagnostics Management

SQL'92

SQL'92 adds a new area of functionality: *diagnostics management*. This standardizes the return codes and completion codes for SQL statements. This may have some security functionality, especially when combined with external procedures.

6 Summary/Recommendations

SQL-compliant DBMS's can be applied to any scenario, no matter what security policy is required. However, not all SQL-compliant DBMS's will be appropriate for every security policy. Many critical features are not specified by SQL; others are specified in one version of SQL but omitted from earlier versions. The systems acquisition phase must begin with a clear and concise statement of the security policy. The exact features required will be a function of that policy.

This section provides a short review of security features that may be found in SQL-compliant DBMS's. These features are classified as unspecified or required. Where required, the version of SQL is specified and a brief summary of the functionality is provided. For unspecified features, the feature is classified as an OS, hardware, or DBMS feature. The remainder of this section discusses the various features and provides guidance regarding their relative importance.

Table 2 summarizes the types of security-relevant controls that might be offered in an SQL-compliant DBMS. Controls are grouped according to security requirements. For each control, a variety of mechanisms is listed.[17] Each control is either required by a version of SQL, or represents unspecified functionality. If the control was a new requirement, the *Status* field will state *required.* If functionality is added to that requirement in later versions of SQL, the *Status* field will say *enhanced.* If the control is probable for SQL3, the *Status* will be *planned.* Note that functionality denoted unspecified or planned may exist in products today; however, implementations will be no-standard.

SQL processors can support a variety of security policy mechanisms. In the area of security policy, the most important decision regards the type of access controls desired. If discretionary controls are desired, SQL'89 does include powerful controls. SQL'92 significantly enhances these controls with the specification of the privilege revocation mechanism. Roles will be a significant enhancement for SQL3. This functionality may be available in SQL processors even before the SQL3 specification becomes stable. Mandatory controls are not specified in any version of SQL but can be supported by SQL implementations.

The SQL integrity constraints are powerful tools for enforcing and maintaining integrity. SQL'89 includes a powerful suite of integrity constraints. SQL'92 does include enhancements such as assertions and domains. These constraints may be used to enforce internal or external consistency constraints.

[17]In some cases, controls map to mechanisms on a one-to-one basis. For example, mandatory access control is basically performed one way. In these cases, the mechanism field is omitted.

Security Requirements	Security Mechanisms: Class	Security Mechanisms: Mechanism	Status In SQL	Where It's Found
Security Policy	DAC	Privileges	required	SQL'89
			enhanced	SQL'92
		Roles	planned	SQL3
	Mandatory Access Control		unspecified	DBMS and OS
	Integrity Constraints	Constraints on Tables	required	SQL'89
			enhanced	SQL'92
		Constraints on Columns	required	SQL'89
			enhanced	SQL'92
	Inference		unspecified	DBMS add-on tools
	Aggregation		unspecified	DBMS add-on tools
	Object Reuse		unspecified	design of OS and DBMS
Accountability	Identification and Authentication	authentication of system session	unspecified	OS dependent
		authentication of DBMS session	unspecified	DBMS implementation dependent
		authentication of each transaction	unspecified	DBMS implementation dependent
	Auditing	Journal generation	unspecified	design of DBMS
		Journal protection	unspecified	design of OS
Assurance	Software Quality	SQL features	unspecified	FIPS conformance tests
		Security features	unspecified	NCSC evaluation
	Fault Tolerance	processor	unspecified	hardware/OS
		disk shadowing	unspecified	hardware/OS
		data mirroring	unspecified	hardware/OS
	Transaction Management		required	SQL'89

Table 2: Security features in SQL standards.

If the system will utilize MAC, design tools should be obtained to limit the threat of aggregation. Whether MAC or DAC policies are envisioned, inference and object reuse are threats to confidentiality. Inference is addressed through add-on tools; object reuse must be addressed within both the DBMS and OS.

Accountability is unspecified in SQL, but the choices are of great importance. Most important is the selection of appropriate authentication points. Should each transaction require re-authentication, or is the session information sufficient? The type of authentication mechanism is also important, but falls outside the SQL specification as well.

Auditing and warning mechanisms are similarly unspecified but required for any reasonably secure system. These mechanisms, especially auditing mechanisms, depend upon the identity previously authenticated. They are of limited value if authentication is weak.

Finally, consider assurance mechanisms. If the system will rely on SQL-specified DAC functionality, SQL conformance testing may be sufficient. If the system will rely upon MAC controls, evaluation may be more applicable.

Fault tolerance is an expensive option, but may be justified if the value of the data is correspondingly high. Disk array technology provides fault tolerance for data storage.

Transaction management features can add assurance that applications are well-behaved, and the database remains in a consistent state. These features add assurance only if they are used consistently and appropriately. If the concept of consistent state is not well understood for the database in question, it will be difficult to use these features appropriately.

References

[ANS89a] Database language - SQL with integrity enhancements. American National Standard X3.135, American National Standards Institute, 1989.

[ANS89b] Database language - embedded SQL. American National Standard X3.168, American National Standards Institute, 1989.

[ANS92] Database language SQL. American National Standard X3.135-1992, American National Standards Institute, 1992.

[Bur89] Rae K. Burns. DBMS integrity and security controls. In *Report of the Invitational Workshop on Data Integrity*. NIST Special Publication 500-168, 1989.

[Cam90] John Campbell. A brief tutorial on trusted database management systems (executive summary). In *13th National Computer Security Conference Proceedings*, 1990.

[CO92] S.J. Cannan and G.A.M. Otten. *SQL - The Standard Handbook*. McGraw-Hill Book Co., Berkshire SL6 2QL England, October 1992.

[DD92] C.J. Date and Hugh Darwen. *A Guide to the SQL Standard*. Addison-Wesley Publishing, Reading, MA 01867 USA, October 1992.

[DJ92] Vinti M. Doshi and Sushil Jajodia. Enforcing entity and referential integrity in multilevel secure databases. In *15th National Computer Security Conference Proceedings*, 1992.

[FIP90] Database language SQL. Federal Information Processing Standard 127-1, National Institute of Standards and Technology, 1990.

[FIP93] Database language SQL. Federal Information Processing Standard 127-2, National Institute of Standards and Technology, June 1993.

[Gal91] Leonard Gallagher. SQL3 support for CALS applications. NISTIR 4494, National Institute of Standards and Technology, February 1991.

[GS92] Leonard Gallagher and Joan Sullivan. Database language SQL: Integrator of CALS data repositories. NISTIR 4902, National Institute of Standards and Technology, September 1992.

[ISO90a] Remote database access - part 1: Generic model. ISO/JTC1/SC21 N4282, Information Processing Systems - Open Systems Interconnect, 1990.

[ISO90b] Remote database access - part 2: SQL specialization. ISO/JTC1/SC21 N4281, Information Processing Systems - Open Systems Interconnect, 1990.

[KGP89] Randy H. Katz, Garth A. Gibson, and David A. Patterson. Disk system architecture for high performance computing. In *Proceedings of the IEEE, Vol. 77, No. 12*. Institute of Electrical and Electronics Engineers, 1989.

[MS92] Jim Melton and Alan Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufman Publishers, San Mateo, CA 94403 USA, October 1992.

[Rob91] Edward Roback. Glossary of computer security terminology. NISTIR 4659, National Institute of Standards and Technology, September 1991.

[SFD92] Linda M. Schlipper, Jarrellann Filsinger, and Vinti M. Doshi. A multilevel secure database management system benchmark. In *15th National Computer Security Conference Proceedings*, 1992.

[TCS85] Trusted computer system evaluation criteria. DOD 5200.28-STD, National Computer Security Center, December 1985.

[TDI91] Trusted database management system interpretation. NCSC-TG 021, National Computer Security Center, April 1991.

[Thu92] Bhavani Thuraisingham. Knowledge-based inference control in a multilevel secure database management system. In *15th National Computer Security Conference Proceedings*, 1992.

[Wag89] Grant Wagner. System services - group 3 report. In *Report of the Invitational Workshop on Data Integrity*. NIST Special Publication 500-168, 1989.