Performance study of a COTS Distributed DBMS adapted for multilevel security

Moses Garuba

Technical Report
RHUL–MA–2004–2
23 July 2004

Royal Holloway
University of London

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
`http://www.rhul.ac.uk/mathematics/techreports`

Abstract

Multilevel secure database management system (MLS/DBMS) products no longer enjoy direct commercial-off-the-shelf (COTS) support. Meanwhile, existing users of these MLS/DBMS products continue to rely on them to satisfy their multilevel security requirements. This calls for a new approach to developing MLS/DBMS systems, one that relies on adapting the features of existing COTS database products rather than depending on the traditional custom design products to provide continuing MLS support.

We advocate fragmentation as a good basis for implementing multilevel security in the new approach because it is well supported in some current COTS database management systems. We implemented a prototype that utilises the inherent advantages of the distribution scheme in distributed databases for controlling access to single-level fragments; this is achieved by augmenting the distribution module of the host distributed DBMS with MLS code such that the clearance of the user making a request is always compared to the classification of the node containing the fragments referenced; requests to unauthorised nodes are simply dropped.

The prototype we implemented was used to instrument a series of experiments to determine the relative performance of the tuple, attribute, and element level fragmentation schemes. Our experiments measured the impact on the front-end and the network when various properties of each scheme, such as the number of tuples, attributes, security levels, and the page size, were varied for a Selection and Join query. We were particularly interested in the relationship between performance degradation and changes in the quantity of these properties. The performance of each scheme was measured in terms of its response time.

The response times for the element level fragmentation scheme increased as the numbers of tuples, attributes, security levels, and the page size were increased, more significantly so than when the number of tuples and attributes were increased. The response times for the attribute level fragmentation scheme was the fastest, suggesting that the performance of the attribute level scheme is superior to the tuple and element level fragmentation schemes. In the context of assurance, this research has also shown that the distribution of fragments based on security level is a more natural approach to implementing security in MLS/DBMS systems, because a multilevel database is analogous to a distributed database based on security level.

Overall, our study finds that the attribute level fragmentation scheme demonstrates better performance than the tuple and element level schemes. The response times (and hence the performance) of the element level fragmentation scheme exhibited the worst performance degradation compared to

the tuple and attribute level schemes.

Performance study of a COTS Distributed DBMS adapted for multilevel security

Moses Garuba
Royal Holloway, University of London

A thesis submitted for the degree of
Doctor of Philosophy

December, 2003

Contents

List of Figures

Acknowledgements

I am sincerely grateful to my thesis advisers: Professor Dieter Gollmann for his time, support, and encouragement beyond the call of duty, and Professor Chris Mitchell for his moral support and counsel, for providing the necessary administrative support, and for proof reading the thesis.

I am also grateful to Commander Kurt Henry of the U.S. Defense Advanced Research Projects Agency (DARPA) and Professor Thomas Keefe of Penn State University for enlightening discussions regarding multilevel database security.

Abstract

Multilevel secure database management system (MLS/DBMS) products no longer enjoy direct commercial-off-the-shelf (COTS) support. Meanwhile, existing users of these MLS/DBMS products continue to rely on them to satisfy their multilevel security requirements. This calls for a new approach to developing MLS/DBMS systems, one that relies on adapting the features of existing COTS database products rather than depending on the traditional custom design products to provide continuing MLS support.

We advocate fragmentation as a good basis for implementing multilevel security in the new approach because it is well supported in some current COTS database management systems. We implemented a prototype that utilises the inherent advantages of the distribution scheme in distributed databases for controlling access to single-level fragments; this is achieved by augmenting the distribution module of the host distributed DBMS with MLS code such that the clearance of the user making a request is always compared to the classification of the node containing the fragments referenced; requests to unauthorised nodes are simply dropped.

The prototype we implemented was used to instrument a series of experiments to determine the relative performance of the tuple, attribute, and element level fragmentation schemes. Our experiments measured the impact on the front-end and the network when various properties of each scheme, such as the number of tuples, attributes, security levels, and the page size, were varied for a Selection and Join query. We were particularly interested in the relationship between performance degradation and changes in the quantity of these properties. The performance of each scheme was measured in terms of its response time.

The response times for the element level fragmentation scheme increased as the numbers of tuples, attributes, security levels, and the page size were increased, more significantly so than when the number of tuples and attributes were increased. The response times for the attribute level fragmentation scheme was the fastest, suggesting that the performance of the attribute level scheme is superior to the tuple and element level fragmentation schemes. In the context of assurance, this research has also shown that the distribution of fragments based on security level is a more natural approach to implementing security in MLS/DBMS systems, because a multilevel database is analogous to a

distributed database based on security level.

Overall, our study finds that the attribute level fragmentation scheme demonstrates better performance than the tuple and element level schemes. The response times (and hence the performance) of the element level fragmentation scheme exhibited the worst performance degradation compared to the tuple and attribute level schemes.

Chapter 1

Introduction

Organisations have long known the importance of restricting access to sensitive information to prevent competitors from learning about plans, new products, or changes in strategies. Confidentiality, integrity, and availability are considered as some of the most important aspects of information assets that must be protected [38]. For most organisations, the database is the primary repository of information; it therefore follows that this asset is the target of many security efforts aimed at protecting it from being compromised. A number of models have been proposed in the past for protecting database management systems; a few product developments and deployments have followed. However, for products based on the multilevel approach, direct support is no longer provided by vendors, even though users have sought it [53]. This thesis addresses this need by proposing and demonstrating the feasibility of an alternative approach to implementing multilevel security, one that exploits the distribution and query processing features of a distributed COTS DBMS.

The next section provides background information about database management systems (DBMSs) and the evolution of computer security models. Section 1.2 elaborates on the problems that motivated this research. Our research aims are described in Section 1.3. The contributions of this research are reviewed in Section 1.4. Section 1.5 provides a guide to the rest of the thesis.

1.1 Background

Database management systems first appeared in the 1960s and have been subject to rapid changes in concepts and technology for over thirty years. A DBMS is a computer system which is responsible for the storage and maintenance of databases [59]. A DBMS is essentially a software system but, in order to make the management of data more efficient, it may contain specialised hardware such as special disk drives that support faster access to the data, and multiprocessors that support parallelism [59]. In addition, DBMSs provide for the safety of information through backup, concurrency, and recovery mechanisms.

These fundamental database concepts are now well defined and understood [35, 59]. Three major DBMS models have been proposed: the Hierarchical Data Model, the Network Data Model, and the Relational Data Model [35, 59]. The *relational model* is a database that is perceived by its users as a collection of tables (and tables only) [38]. It has become the predominant model, and it is the model that will be used in this thesis. We will return to the subject of DBMSs with a more detailed description later in Chapter 2. Another important concept which is central to our discourse, and that evolved in parallel with the relational data model, is computer security.

A number of definitions have been proposed for computer security, but most of the definitions embody the confidentiality, integrity, and availability aspects of protecting information. However, for the purposes of this thesis, computer security is defined as *the prevention and detection of unauthorised actions by users of a computer system* [38]. Most of the early work in computer security focused on military security. The U.S. Department of Defense funded the initial efforts because of its need to control the disclosure of information which could damage U.S. national security interests [30, 31]. A multilevel security hierarchy with four levels of increasing sensitivity was developed. The levels from lowest to highest are: Unclassified (U), Confidential (C), Secret (S), and Top Secret (TS), ignoring for now additional categories within levels such as (Secret [Nuclear]) versus (Secret [Conventional]) [26, 72]. The four level system and its categories was designed to capture only a very specific "military" security policy. Later, we will consider systems that use different sets of security levels. Data in a multilevel secure computer system is required to be labelled with its security classification. Users of the data are required to have the appropriate clearance level to access the data in the system. Secure computer systems compare the clearance of the user to the classification of the data and mediate the access of the users to the data in accordance with specified security rules. For example, a basic security rule incorporated in the *Bell-LaPadula disclosure model (BLP)* [7, 8] is that no user can read any data with a greater classification than the user's clearance (e.g., a Secret user cannot have access to Top Secret data). Since both clearances of users and classifications of data are constructed in the same manner, they will both be referred to using the term level.

The BLP model [7, 8] was the first mathematical specification of a multilevel security policy; it was central to the development of basic computer security standards and laid the groundwork for a number of later security models, and their application in U.S. government security standards. The model is described in terms of objects (a file, data, memory, I/O devices, etc.), subjects (users or a process acting on behalf of a user) and their corresponding security levels. The goal of preventing unauthorised disclosure can be satisfied if a user, or a subject acting on behalf of a user, is restricted to those objects whose level is dominated by the user. The BLP is the basis of the multilevel security policy in DBMSs with which we are concerned here.

Major research efforts were undertaken in the early 1980's to incorporate multilevel security principles into relational database management systems [30]. The latter effort, known as multilevel secure database management systems, generated much progress in the years that followed. A *multilevel secure DBMS (MLS/DBMS)* extends the classical relational data model to include security classification attributes and further integrity constraints. We will postpone a detailed treatment of these concepts until later in Chapter 2. Database systems based on the classical relational data model are used in a wide variety of application areas, including banking systems, reservation systems, and military intelligence systems.

As a result of the diversity of application domains for database systems, a number of different security models and techniques have been proposed to counter the various threats against security. A security model is implemented on the basis of a specific access control policy; this could be a *discretionary access control (DAC)* or a *mandatory access control (MAC)* policy. With a DAC policy, users at their discretion can specify to the system who can access their files. In the case of secure database systems for use in applications that call for multilevel security (such as those of the U.S. Department of Defense where data security is of primary importance), discretionary access controls are not adequate, and more stringent restrictions on the sharing of information are required. A MAC policy provides the required restrictions. In contrast to a DAC policy, a MAC policy is a "system enforced" policy that restricts access to objects based on the classification of the object and the clearance of the user to access information at that sensitivity level; this type of policy forms the basis of the work described in this thesis. Discretionary and mandatory access controls are described in more detail in Chapter 2.

Most systems supporting DAC store access rules in an access control matrix. In its simplest form the rows of the matrix represent subjects, the columns represent the objects, and the intersection of a row and a column contains the access type or types for which a subject is authorised with respect to the object. The access matrix model as a basis for discretionary access controls was formulated by Lampson [57] and subsequently refined by Denning [25]. Other models that support DAC include the *Take-Grant model* [49] and the *Action-Entity model* [14]. A formal description of DAC will be presented in Chapter 2.

MAC policies address a higher level of threat than discretionary policies because, in addition to controlling access to data, they can also be used to control the flow of data. MAC requirements are often stated using the BLP model, and are formalised using two rules. The first, the *simple property*, protects the database information from unauthorised disclosure, and the second, the **-property*, prevents the unauthorised downgrading of data by restricting information flow from high security levels to low security levels. Some of the models that support MAC include the *SeaView model* [28], the *Jajodia-Sandhu model* [48], the *Smith-Winslett model* [75], and the *Multilevel Relational model* [20]. These models will be described in more detail in Chapter 3.

Providing a database system with multilevel security, or bringing a model to life as a prototype or a commercial product, is fraught with difficulties. A number of approaches have been proposed; one such approach advocates designing the security mechanisms into the database system itself and then trusting the database to enforce the security policy [2]. In practice this results in a low assurance system; that is, the separation between security policy and enforcement is weak [2]. This kind of database system is useful, but cannot interconnect the diverse population of users expected for the new information infrastructure. The reason for the low assurance is the complexity and size of modern database systems. Less straightforward but more effective approaches use a reference monitor to enforce the security policy. The reference monitor is evaluated for high assurance and the database system is designed to function under the security constraints enforced by the reference monitor.

In a search for a more effective solution, the Multilevel Data Management Security Summer Study [1] recommended three approaches to solving the multilevel database security problem. The three approaches are: integrity lock, kernelized, and replicated. The integrity lock approach, also known as the *spray paint architecture*, uses a trusted front-end, a single untrusted back-end database system, and encryption techniques to protect data. In the kernelized approach, the multilevel database is partitioned into single-level databases, which are then stored separately under the control of a trusted security kernel that enforces a MAC policy. In addition, there is a separate DBMS for each security level. In the replicated approach, a separate DBMS is used to manage data classified at or below each security level. Unlike the kernelized approach, each database in the replicated approach maintains data at its level and below. We will discuss all three approaches in more detail in Chapter 3.

Building on the above models, a number of development efforts to build MLS prototypes and high assurance commercial MLS/DBMS products were initiated. The SeaView model was developed as a prototype MLS/DBMS that uses viewlike mechanisms for access control; the *Lock Data Views* (LDV) prototype, the *A1 Secure DBMS* (ASD) prototype, the *SWORD* prototype, and the *Purple Penelope* demonstration system followed afterwards. A number of commercial products were also deployed as a result of the pioneering efforts of database vendors such as Oracle, Ingres, Sybase, and Informix, but these systems are no longer being supported.

In order to be certified at a particular level of security, the prototypes and products must be evaluated (and tested) in an adversarial manner. The originally used criteria for evaluating secure computer systems are contained in the U.S. Department of Defense "Trusted Computer System Evaluation Criteria" (TCSEC), also known as the Orange Book [7, 8, 64], although this has since been replaced by the Common Criteria [67]. For high assurance systems that must provide a strong separation between security policy and enforcement, the protection mechanisms must be simple and easy to evaluate. Protection critical components must contain only protection mechanisms and must mediate every access by each user. In general, these systems are more carefully engineered and crafted than most computer systems. Protection critical components must be scrutinized in an adversarial way in order to ensure that malicious code is not present in the system. The Orange Book proposed three protection critical concepts, the *Reference monitor*, the *Security kernel*, and the *Trusted Computing Base* (TCB) as the central notions of a trusted system [38, 64]. The reference monitor is an access control concept that refers to an abstract machine that mediates all accesses to objects by subjects [64]. The security kernel is the hardware, firmware, and software elements of a trusted computing base that implements the reference monitor concept. It must mediate all accesses, be protected from modification, and be verifiable as correct [64]. The TCB refers to the totality of protection mechanisms within a computer system – including hardware, firmware, and software – the combination of which is responsible for enforcing a security policy [64].

1.2 Problem statement

The MLS/DBMS products that were developed and distributed in the eighties and nineties by vendors such as Ingres, Oracle, Sybase, and Informix are no longer being supported. The decline in vendor support has been attributed mainly to a scaling back of research and development grants by the U.S. Department of Defense, the catalyst behind many development efforts. As a consequence of the withdrawal of government funding, many of the vendors were unwilling to continue supporting these products, already considered to be too complex and unprofitable.

The number of existing users of COTS MLS/DBMS products, however, remains significant; these users have invariably found themselves in a precarious situation with limited or no vendor support. Some of these users sought alternatives to MLS-based systems; others continued to operate on MLS policies. There are no new products currently on the market to fill the void left by this absence of support. These existing users will need COTS MLS/DBMS support for as long as they continue to operate using MLS policies. This calls for new COTS MLS/DBMS initiatives that meet that need.

A multilevel database consisting of relations (tables), which are fragmented based on security level, is logically the same as a distributed database. However, none of the existing MLS/DBMSs use the inherent advantages of distributed DBMSs to access or to provide security to single-level fragments. These distributed DBMS techniques support the fragmentation of relations and the distribution of their fragments at the tuple, attribute, and element levels of granularity. Although we suggest fragmentation as a good basis for implementing MLS security, the relative performance of the tuple, attribute, and element-level fragmentation schemes needs to be determined through investigation. The benchmark information that this investigation provides is necessary to inform organisations that are interested in implementing an MLS/DBMS based on our approach.

While MLS/DBMSs have been widely criticised for their inefficiency and complexity [53], we are only aware of one benchmark study that has sought to investigate their performance. The study by Thuraisingham and Kamon [77] was based on the distributed architecture approach recommended by the Multilevel Data Management Security Summer Study [1, 6]. The purpose of the benchmark study was to investigate the performance of query processing algorithms in this particular implementation. The study did not investigate the performance of the fragmentation schemes that are of interest to our research as it was based on a trusted DBMS that supported only one type of partitioning scheme. Furthermore, only a limited number of variables were subject to observations to determine their impact on the system.

1.3 Aims of this research

The work contained in this thesis is motivated by the need to demonstrate the feasibility of using a cluster of machines and a COTS distributed DBMS software to implement multilevel security. This need became apparent because of the absence of COTS support for existing users of MLS/DBMS products, and the lack of an alternative system that satisfies the security needs of these users. To that end, we will draw on the query processing techniques and fragmentation features developed for distributed databases to more efficiently enforce security in MLS/DBMSs which are implemented as kernelized architectures. We demonstrate this concept by augmenting the Stargres distributed database management system[1] with multilevel code to deliver security at three levels of granularity: tuple, attribute, and element.

We have adapted Stargres, based on the kernelized approach, to provide multilevel database security. Henceforth, we refer to this adapted distributed DBMS as the Multilevel Stargres (MST) prototype. MST enforces a strong MAC protection policy and provides the performance and full data management capabilities of conventional industry-standard database management systems. It also uses physical separation as its primary protection mechanism. The MST prototype consists of a framework for mandatory access security, that includes local DBMSs, and a Distributed server.

In our benchmark study, we investigated the performance of our proposed kernelized approach to multilevel security to determine how it performs under the tuple, attribute, and element-level fragmentation schemes for the Select-All and Join queries. Our experiments

[1]Stargres is a product of Johns Hopkins University Applied Physics Laboratory, Maryland.

compared the performance impact of varying such diverse variables as tuples, attributes, security levels, number of users, and the page size. It is important to determine how a variation in the quantity of each variable impacts the performance of each fragmentation scheme, so as to identify potential bottlenecks, and to suggest the combination of variable quantities that will optimise the performance of each scheme.

1.4 Contributions of this research

The work described in this thesis builds upon prior research in a number of different fields, and is up to date as of Spring 2000 — the date of its original submission. There have been further developments in MLS/DBMSs research since Spring 2000 although we have not attempted to cover them here.

This section summarises how our research contributes to this body of knowledge. Our work:

- demonstrates that the query processing and distribution techniques developed in distributed databases can be used to implement security in MLS/DBMSs which are implemented as kernelized architectures. This concept was elucidated (or brought to life) by adapting a COTS distributed database management system to provide multilevel security.
- proposes a conceptual MLS/DBMS model that is largely based on models oriented towards representing data 'truthfully', e.g., the SWORD approach rather than those that are oriented towards representing data 'honestly', e.g., the Jajodia-Sandhu model. Although the model uses the Jajodia-Sandhu definition of a multilevel relational model, its operational semantics is mostly based on the SWORD approach.
- demonstrates the feasibility of adapting a COTS DBMS to provide multilevel security; it shows that the same techniques used in the adaptation can also be utilised by subscribers to MLS policy who no longer enjoy direct vendor support.
- demonstrates that the adaptation of a COTS DBMS can be achieved solely by identifying the critical modules that handle data objects and administrative/user requests, and then augmenting them with MLS security code. This process need not be as complex as the implementation of an MLS/DBMS from scratch.
- investigates the relative performance of the three fragmentation schemes supported by our prototype by conducting a number of experiments using variables such as tuples, attributes, security levels, number of users, and the page size, for a Select-All and Join query.

- shows, through our observations, that the performance of the attribute-level fragmentation scheme is superior to the tuple and element-level fragmentation schemes, and that the element-level scheme suffers from the worst performance degradation.

We will return to this list of contributions at the conclusion of this thesis and elaborate on each contribution.

1.5 Contents of the thesis

In this chapter, we have described the background to our research, outlined the subject matter of this thesis, provided an overview of multilevel security, and described the scope of our work. Chapter 2 reviews the basic concepts used throughout the thesis.

In Chapter 3, we review some previous efforts at protecting data from unauthorised disclosure, and the state-of-the-art in multilevel security focusing on MLS models, architectures, prototypes, and product development.

In Chapter 4, we present a conceptual MLS/DBMS model that is oriented towards representing data 'truthfully', e.g., the SWORD approach, but that also incorporates features from approaches that represent data 'honestly', e.g., the Jajodia-Sandhu model. We also describe the axioms and salient features of the proposed model.

In Chapter 5, we present the architecture of the Stargres distributed database management system, including a description of its constituent facilities and modules, and how they cooperate in processing queries.

In Chapter 6, we describe the technical details of our proposed MST prototype, focusing on the Distributed server and the modules that were adapted to provide MLS functionality. We also discuss the hardware configuration and usage of the prototype.

In Chapter 7, we investigate the performance of the MST prototype's three fragmentation schemes for a Select-All query and a Join query. In our study, we examine the impact of varying properties such as the number of tuples, number of attributes, number of security levels, and the page size, on the front-end and the network.

In Chapter 8, we summarise our contributions and point to further areas of research that can be developed to build upon the work done in this thesis.

Chapter 2

Review of Basic Concepts

2.1 Introduction

This chapter introduces a number of basic concepts that will be used throughout this thesis. The discussion consists of five sections. Section 2.2 describes database systems in general, focusing on their constituent software components. Section 2.3 presents the relational data model and some of its Structured Query Language (SQL) operators. In section 2.4 we present discretionary access control and mandatory access control, including ordered sets, lattices, and the Bell-LaPadula model. The multilevel relational model, an extension of the standard relational model, is described in section 2.5. Section 2.6 highlights some important features of distributed database systems that are important to the discussion of the COTS distributed DBMS architecture introduced in Chapter 5, these features include relation fragmentation and query processing. Section 2.7 summarises the chapter.

2.2 Database systems

A *database system* is a computerised system to keep track of information. The information in the database system consists of both data and information about the data *(metadata)* such as the relationship of one data item to another. A database system can be viewed as having four parts [24]: data, users, hardware, and software.

Data: These are raw information that a business, government agency, or some other group of individuals collect to fulfil their goals or missions. Individual data items are collected into sets of related data items called records. A collection of interrelated records is called a database.

Users: The various individuals and groups of people who use information are defined as users.

Hardware: Hardware typically consists of physical devices such as disks, printers, I/O devices and the processor itself, with its associated memory.

Software: The interface between the physical data and the user of the data is the *Database Management System* (DBMS). The DBMS is essentially a software system, but may contain specialised hardware in order to make the management of data more efficient [59]. Such hardware may include special disk drives that support fast access to the data, and multiprocessors that support parallelism [59].

The DBMS provides the various users of the database with different ways of looking at the data depending on their uses of the data. These different ways of looking at the data represent distinct levels of abstraction of the underlying data: the *Physical level* is the lowest level of abstraction. Typically, this level is of interest to the designers of the DBMS software who are concerned with how the data are physically stored (e.g., the disk address of a record). The *Conceptual level* is the middle level of abstraction, concerned with describing what data are in the database and the relationships between data items. This level is of interest to Database Administrators and Security Administrators. The *View level* is the highest level of abstraction. Typically, this level is the way the end-user sees the data. Each end-user can have a tailor-made view of the data which are of interest to him or her. This view of the data does not require the user to know or understand the internal characteristics of the data (such as how they are represented or stored).

Database models permit differentiation between the description of the database, which is specified in the schema, and the collected contents or values of the data items in the database at any particular time, which is called the *instance*. The database schema is specified using a *data definition language* (DDL). The manipulation of actual data in the database (inserting, deleting, updating, or retrieving data values) is accomplished using a *data manipulation language* (DML).

The DBMS uses a complex set of software components in order to perform its function. These components include the *data manager* which provides an interface to physical data stored in the database, the *query processor* which translates the query language into instructions for the data manager, the *data manipulation language precompiler* which translates DML statements from applications programs to host language calls, and works with the query processor, and the *data definition language compiler* which translates DDL statements into tables of metadata. Metadata are stored in the *data dictionary*, which includes the structure of the database *or the schema* and *data integrity constraints* (e.g., age must be numeric and between 0 and 120), and *security constraints* (data item x is secret).

The *relational data model* is described next. This is the model that forms the basis of the work described in this thesis.

2.3 The relational data model

We stated in Chapter 1 that a database is represented by the relational model as a collection of tables (see figure 2.1). More importantly, the model is directly related to the mathematical concept of a relation; it is comprised of:

1. **a structural part.** A database schema is a collection of relation schemas and a database is a collection of relations.
2. **an integrity part.** Primary keys and foreign keys.
3. **a manipulative part.** Relational algebra and relational calculus.

Formally, a relation R is a subset of $D_1 \times \ldots \times D_n$ where $D_1, \ldots, D_n$ are the *domains* of n *attributes* $A_1, \ldots, A_n$. The elements in the relation are n-tuples $(v_1, \ldots, v_n)$ with $v_i \in D_i$, i.e., the value of the ith attribute has to be an element from D_i. The elements in a tuple are often called *fields*. When a field does not contain any value, we represent this by entering a special *null value* in this position, meaning ‘there is no entry’ rather than ‘entry is unknown’.

DN	**Dept_Name**	**Manpower**	**Dept_Location**	**Manager**
D10	Records	43	241 East Annex	Cantor
D11	Obstetrics	13	601 Main Building	Baker
D12	Laboratory	8	202 Drew Building	Ebert
D13	Geriatrics	null	B16 Basement	Xavier

Figure 2.1: A *department* relation (or table).

2.3.1 Integrity constraints

Integrity constraints restrict the set of theoretically possible tuples to a set that is practically meaningful. Let X and Y denote sets of one or more of the attributes A_i in a relation schema. We say Y is functionally dependent on X, written $X \rightarrow Y$, if and only if it is not possible to have two tuples with the same value for all the attributes in X but different values for any of the attributes in Y. Functional dependencies represent the basis for most integrity constraints in the relational model. As not all possible relations are meaningful in an application, only those that satisfy certain integrity constraints are considered. From the large set of proposed integrity constraints, *entity integrity* and *referential integrity* are of major importance.

- **the entity integrity constraint** states that each tuple must be uniquely identified by a key and a key attribute cannot be null;

- **the referential integrity constraint** states that an n-tuple in one relation that refers to another relation must refer to an existing n-tuple in that relation; this is expressed by means of foreign keys.

These two rules are application-independent, and must be valid in each relational database. In addition, many application-dependent semantic constraints may exist.

A *candidate key* of a relation schema R is a minimal set of attributes on which all other attributes of R are functionally dependent. The *primary key* of a relation schema R is one of its candidate keys that has been specifically designated as such. A *foreign key* of a relation schema R is a set of attributes in a relation schema that forms a primary key of another relation. For a more detailed treatment of the relational model including its core integrity properties, see [35, 59]; for now, we consider how queries against relations are expressed.

2.3.2 Structured query language

The relational model has resulted in query languages such as the *relational algebra*, which is the procedural query language for the relational model, and the *relational calculus*, which is the declarative counterpart of the relational algebra and is based on first-order predicate calculus [59]. Relational calculus provides the theoretical underpinning of the *Structured Query Language (SQL)*, the commercial relational query language used in most DBMSs [59]. A description of the relational algebra, relational calculus, and SQL operations may be found in many references discussing the relational model, see [18, 24, 59].

SQL was developed during the 1970's at IBM as part of the System R[1] project and was standardised during the 1980's by the International Standards Organisation (ISO). In the following, we introduce a subset of SQL operations that will be used later in this thesis, including the Retrieve, Insert, Update, and Delete operations. The operations will be illustrated using the Department and Nurse relations shown in Appendix A.

RETRIEVE statement

SELECT *attribute_list*
FROM *table_list*
WHERE *selection_condition*

The SELECT statement selects data from a table and stores it in a result-set. Only those tuples which satisfy the selection_condition are included in the result-set which has the same schema as the original table. The operations allowed in a *selection_ condition* include =, !=, <, >, and LIKE.

[1]System R is a relational DBMS prototype developed by IBM at the San Jose Research Laboratory in California.

Example: A retrieval from a single table, to find the Dept_Name with a DN value of D12 and its corresponding Manpower value in the Department table.

SELECT Dept_Name, Manpower
FROM Department
WHERE DN = ‘D12’;

Example: A retrieval from two tables (JOIN), to find the names of all departments, their corresponding manpower, and the specialties of their nurses from the Department and Nurse tables where the DN values in the Department table corresponds to the DN values in the Nurse table, and the manager is Cantor.

SELECT Department.Dept_Name, Department.Manpower, Nurse.N_Specialty
FROM Department, Nurse
WHERE Department.DN = Nurse.DN
AND Department.Manager = ‘Cantor’;

INSERT statement

INSERT INTO *table_name* $(attribute1, attribute2, \ldots)$
VALUES $(value1, value2, \ldots)$

The INSERT statement inserts new tuples in a table. The attribute names may be omitted if both their number, and the order in which the values to be inserted appear, agree with the original table.

Example: To INSERT a tuple in the Department table.

INSERT INTO Department (DN, Dept_Name, Manpower, Dept_Location, Manager)
VALUES (‘D30’, ‘Psychiatry’, 16, ‘105 South Wing’, ‘Smith’);

UPDATE statement

UPDATE *table_name*
SET *attribute_name* = new_value
WHERE *selection_condition*

The UPDATE statement sets the attribute_name specified to the corresponding value for all those tuples in the specified relation that satisfy the *selection_ condition.* If no condition is given, all tuples are updated.

Example: To UPDATE an attribute in the Department table.

UPDATE Department
SET Manpower = 16
WHERE DN = 'D12';

DELETE statement

DELETE FROM *table_name*
WHERE *selection_condition*

The DELETE statement deletes tuples in a table. If the *selection_ condition* is specified, all the tuples for which this condition holds are deleted. An unqualified deletion will empty the table (without destroying its structure).

Example: To DELETE a tuple from the Department table.

DELETE FROM Department
WHERE DN = 'D12';

2.4 Access control

As the security of the relational database model is the main theme of this thesis, this section contains a review of some of the basic notions of access control. Access control ensures that all direct accesses to database objects occur only according to the models and rules fixed by protection policies [50]. Earlier in Chapter 1, we introduced two types of access control policies, namely DAC and MAC. These access control policies provide different levels of protection to the relations in a system, they are based on the concept of a set of subjects S (e.g., users, groups of users, or transactions operating on behalf of users), to whom we grant or deny access, a set of objects O (e.g., relations, tuples), to which access is either granted or denied, the set of access operations $A = \{$read, write, append, execute, grant, delegate, revoke$\}$, a set of predicates P, used to represent content-based access (for DAC), and a set L of security levels with a partial ordering $\leq$ (for MAC) (see subsection 2.4.2) [7, 38].

2.4.1 Discretionary access control (DAC)

In DAC, a predicate $p \in P$ defines the permissions of subject $s \in S$ on object $o \in O$. The tuple $< o, s, a, p >$ is called an *access rule* and a function f is defined to determine if an authorisation $f(o, s, a, p)$ is valid or not:

$$f : O \times S \times A \times P \rightarrow \{\text{True, False}\}.$$

For any $< o, s, a, p >$, if $f(o, s, a, p)$ is True, subject s has authorisation a to access object o within the range defined by predicate p. An important property of discretionary security models is the support of the principle of *delegation of rights* where a right is the (o, a, p)-portion of the access rule. A subject s_i who holds the access right (o, a, p) may be allowed to delegate that right to another subject $s_j (i \neq j)$. This flexibility to delegate (grant) access rights in DAC makes it a popular choice for implementing commercial and industrial security policies. A more detailed treatment of DAC can be found in [15, 38].

2.4.2 Mandatory access control (MAC)

While DAC is concerned with controlling access to objects, MAC is in addition concerned with the flow of information between different security levels in a system. MAC requires that objects and subjects are assigned to certain security levels represented by a label. The label for an object is called its classification and a label for a user is called its clearance. MAC makes access control decisions by comparing these labels on the basis of rules described in subsection 2.4.2.3. Security labels form a partially ordered set (and often a lattice).

2.4.2.1 Ordered sets

In the following definition of a partially ordered set, let L be a set. A *partial* order on L is a binary relation, $\leq$, on L such that for all $a, b, c \in L$

- $a \leq a$,
- $a \leq b$ and $b \leq a$ imply that a = b,
- $a \leq b$ and $b \leq c$ imply that $a \leq c$.

These three conditions are referred to as *reflexivity*, *antisymmetry* and *transitivity*, respectively. A set L equipped with a partial order is called a *partially ordered set.*

2.4.2.2 Lattices

A *lattice* $(L,\leq)$ consists of a set L and a partial ordering $\leq$, with the property that, for every two elements a, $b \in L$ there exists a *least upper bound* $u \in L$ and a *greatest lower bound* $l \in L$, i.e.,

- $a \leq u$, $b \leq u$, and for all $v \in L$: $(a \leq v \wedge b \leq v) \Rightarrow (u \leq v)$
- $l \leq a$, $l \leq b$, and for all $k \in L$: $(k \leq a \wedge k \leq b) \Rightarrow (k \leq l)$

In MLS policy, if $a \leq b$, we say that a is dominated by b. If there exists a security level that is dominated by all other levels, it is called *System-low*. If there exists a security level that dominates all other levels, it is called *System-high*. Typical examples of lattices are illustrated in figure 2.2. Figure 2.2(A) shows a totally ordered set of security levels where Top Secret dominates Secret, Confidential, and Unclassified; Secret dominates Confidential, and Unclassified; and Confidential dominates Unclassified. Figure 2.2(B) shows a partially ordered set of security levels proposed for firewalls where System high dominates inside, outside, and System low, inside and outside dominate System low, and inside is incomparable to outside.

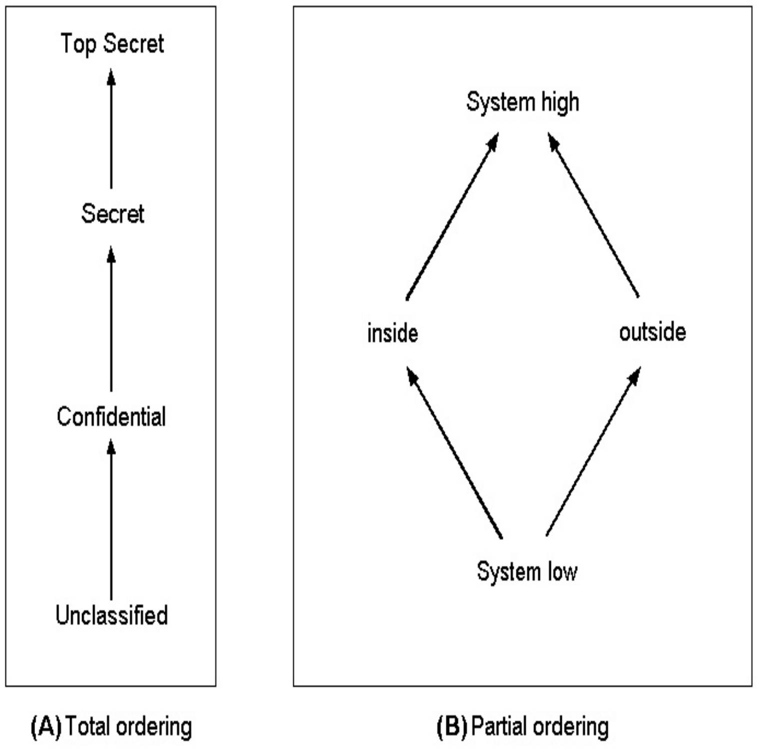

Figure 2.2: Security levels in total order and security levels in partial order showing the dominates $\leq$ relationship between elements.

2.4.2.3 Bell-LaPadula model (BLP)

The BLP model is a state machine model capturing the confidentiality aspects of access control, where access to objects is controlled by a strict set of rules that are enforced by the system. Access permissions are defined both through an access control matrix and through security levels represented by labels. The downwards flow of information from a high security level to a low security level is prohibited by the model when a subject observes or alters an object. The model enforces a *multilevel security (MLS)* policy [7, 8].

We present a formal description of the BLP model using the set of objects, subjects, access operations, and partially ordered security levels. The access operations of the BLP model are: execute, read, append, and write. The current states will be described in these terms. This leads to a state set $B \times \mathcal{M} \times F$, where:

- $B = \mathcal{P}(S \times O \times A)$ is the class of all possible sets of access operations currently permissible within the system. An element $b \in B$ is a collection of tuples (s, o, a), indicating that subject s currently performs operation a on object o.
- $\mathcal{M}$ is the set of access permission matrices $M = (M_{so})_{s \in S, o \in O}$.
- $F \subset L^S \times L^S \times L^O$ is the set of security level assignments. An element $f \in F$ is a triple (f_s, f_c, f_o), where:

  - $f_s : S \rightarrow L$ gives the *maximal security level* each subject can have,
  - $f_c : S \rightarrow L$ gives the *current security level* of each subject,
  - $f_o : O \rightarrow L$ gives the *classification* of all objects.

The current level of a subject cannot be higher than its maximal level, hence $f_c \leq f_s$. The reason for introducing f_c will become clear as this discussion continues. The maximal security level is sometimes called the subject's clearance. Other sources use clearance only to denote the security level of users. Defining the state set is the major issue in BLP. We do not have to describe inputs, outputs, or the precise structure of state transitions, to give the BLP security properties.

Security policies

BLP defines security as the property of states. Multilevel security policies allow a subject to read an object only if the subject's security level dominates the object's classification. These multilevel security policies are also called *mandatory security policies*. The BLP model defines two properties (rules) for mediating the access of subjects to objects; these properties are the simple security property rule (ss-property) and the *-property. Both properties must hold in order for security to be maintained [7, 8, 58]:

> **ss-property.** A state (b, M, f) satisfies the *ss*-property, if for each element $(s, o, a) \in b$ where the access operation a is *read*, the security level of the subject s dominates the classification of the object o, i.e., $f_o(o) \leq f_s(s)$. This property captures the traditional *No Read-up (NRU) security policies.*

The ss-property is, however, not sufficient to prevent a low-level subject from reading the contents of a high-level object. It could create a high-level Trojan horse which reads the high-level object and copies it (writes its contents) into a low-level object. Thus, we also have to control write access through the *-property.

***-property.** A state (b, M, f) satisfies the *-property, if for each element $(s, o, a) \in b$ where the access operation a is *append*, the current level of the subject s is dominated by the classification of the object o, i.e., $f_c(s) \leq f_o(o)$. This is a *No Write Down (NWD) security policy.* Furthermore, if there exists an element $(s, o, a) \in b$ where the access operation a is *append* or *write*, then we must have $f_o(o') \leq f_o(o)$ for all objects o' with $(s, o', a') \in b$ and a' is *read* or *write*.

This definition immediately implies that a high level subject cannot send information to a low-level subject. There are two ways to escape from this restriction:

- Temporarily downgrade a high level subject. This is the reason for introducing the current security level f_c.
- Identify a set of subjects that are permitted to violate the *-property. These subjects are called *trusted subjects*.

The first approach assumes that a subject forgets all it knew at a higher security level the moment it is downgraded. This looks implausible if you view subjects as human beings, but BLP is about modelling computers. There, subjects (processes) have no memory of their own. The only thing they 'know' are the contents of the objects (files) they are allowed to observe. In this situation, a temporary downgrade indeed solves the problem.

In a second interpretation, f_s specifies a user's clearance. Users are allowed to login below their clearance and f_c indicates at which level a user has actually logged in.

When adopting the second approach, the *-property only has to hold for subjects which are not trusted. By definition, a *trusted subject* may violate the security policy. Although it has the ability to violate the security policy of the system, it is trusted not to actually do so.

Discretionary access control in BLP is also expressed by an access control matrix and captured by the *discretionary security property (ds-property).*

ds-property. A state (b, M, f) satisfies the *ds*-property, if for each element of $(s, o, a) \in b$ we have $a \in M_{so}$.

A state is called *secure* if all three security properties are satisfied.

The Basic Security Theorem

A transition from state $v_1 = (b_1, M_1, f_1)$ to state $v_2 = (b_2, M_2, f_2)$ is said to be secure, if both v_1 and v_2 are secure. To see which checks have to be performed to determine whether the new state is secure, consider, for example, the ss-property. The state transition *preserves* the ss-property if and only if:

1. each $(s, o, a) \in b_2 \backslash b_1$ satisfies the ss-property with respect to f_2; ($b_2 \backslash b_1$ denotes the set difference between b_2 and b_1) and
2. if $(s, o, a) \in b_1$ does not satisfy the ss-property with respect to f_2, then $(s, o, a) \notin b_2$.

Preservation of the *-property and of the ds-property can be described in a similar way. We are now in a position to state an important property of the BLP model.

Basic security theorem. If all state transitions in a system are secure and if the initial state of the system is secure, then every subsequent state will also be secure, no matter which input occurs.

A formal proof of this theorem would proceed by induction over the length of input sequences. The proof would build on the fact that each state transition preserves security but would not refer to the specific BLP security properties. In practice, the basic security theorem limits the effort needed to verify the security of a system. You are allowed to check each state transition individually to show that it preserves security and you have to identify a secure initial state. As long as a system starts in this secure initial state, it will remain secure.

Tranquility

McLean [62] triggered a heated discussion in 1985 about the value of the BLP model by putting forward a system that contained a state transition, which

- downgraded all subjects to the lowest security level,
- downgraded all objects to the lowest security level,
- entered all access rights in all positions of the access control matrix M.

The state reached by this transition is secure according to the definitions of BLP. Should such a state be regarded as secure? As BLP says it is secure, does BLP then capture security correctly? There are two opinions:

- **The case against BLP (McLean):** Intuitively, a system that can be brought into a state where everyone is allowed to read everything is not secure. Therefore, BLP has to be improved.
- **The case for BLP (Bell):** If the user requirements call for such a state transition, then it should be allowed in the security model. If it is not required, then it should not be implemented. This is not a problem of BLP but a problem of correctly capturing the security requirements.

At the root of this disagreement is a state transition that changes access rights. Such changes are certainly possible within the general framework of BLP but the originators of the model were really contemplating systems where access rights are fixed. The property that security levels and access rights never change is called *tranquility.* Operations that do not change access rights are called *tranquil.*

Aspects and limitations of BLP

Although BLP is an important security model and has played an important role in the design of secure systems, it does not cover all aspects of security. I has been criticised for: 1) only dealing with confidentiality without addressing integrity, 2) not addressing the management of access control, 3) containing covert channels.

The absence of integrity policies in BLP gave rise to an issue referred to as *blind writes.* Blind writes has been shown to introduce integrity problems in systems complying with the BLP model's NRU and NWD rules. This situation arises when a user is allowed to insert, modify, or delete a record, but is also not allowed to observe the effect of these operations. For example, if a user logged-in at System-low writes a record at System-high, it is generally safe computing practice for the user to observe the contents of the record being modified to make sure that the operation succeeded; however, with BLP, a user at System-low cannot observe the effects of changes at System-high. Blind write raises concerns because the same user deemed unsuitable for viewing a record is permitted to make arbitrary changes to that record. This may cause integrity problems that can only be dealt with by imposing requirements that may change the BLP rules. For example, prohibiting a no write up rule and allowing only writes to records that are at the same security level as the user will restrict the model and shift its focus from disclosure to a combination of integrity and disclosure. Because of this potential integrity problem, few developers have opted to implement the BLP rules as they are written; many have opted instead for a modified version of the rules. Modified versions of the BLP rules have a tendency to be more restrictive, such as disallowing blind writes so that users can only write records at their own security level.

2.5 The multilevel relational model

In the multilevel relational model, relations, tuples, attributes, or elements are assigned security classifications. This thesis uses the definition of a multilevel relation captured by the Jajodia-Sandhu model [47]. The model is based on the security classifications introduced in the BLP model, and it formally defines a multilevel relation as consisting of the following two parts:

1. a *state-independent multilevel relation schema* $R(A_1, C_1, ..., A_n, C_n, TC)$, where each

A_i is a data attribute defined over domain D_i, each C_i is a classification attribute for A_i, and TC is the tuple-class attribute.

2. a collection of *state-dependent relation instances* $R_c(A_1, C_1, ..., A_n, C_n, TC)$, where one such instance exists for each access class c from the set of security levels. Each instance is a set of distinct tuples of the form $(a_1, c_1, ..., a_n, c_n, tc)$ where tc is the least upper bound of the classes of the attributes in the tuple, and tc indicates the lowest user level that can see the tuple. The instance of a relation at a given access class represents the version of the relation at that class [47, 50]. Basically, each element $t[A_i]$ in a tuple t is visible in instances at access class $t[C_i]$ or higher; $t[A_i]$ is replaced by a null value in an instance at a lower access class.

A *single-level relation* is a relation whose elements have the same security classification.

2.5.1 MLS database integrity constraints

In order to meet the requirements of MLS databases, the two core integrity constraints of the relational model described earlier were adapted, and two further constraints introduced. In the standard relational model, a key is derived by using the concept of functional dependencies. In the MLS relational model such a key is called the *apparent key* (AK) which we assume is a user-specified primary key consisting of a subset of the data attributes A_i [46]. The notion of apparent key is also discussed by Castano et al. in [15].

MLS integrity property 1: *Entity integrity.*

Let AK be the apparent key of a relation R. A multilevel relation R satisfies entity integrity if, and only if, for all instances R_c of R and $t \in R_c$

1. $A_i \in AK \Rightarrow t[A_i] \neq null$,
2. $A_i, A_j \in AK \Rightarrow t[C_i] = t[C_j]$, i.e., AK is uniformly classified, and
3. $A_i \notin AK \Rightarrow t[C_i] \geq t[C_{AK}]$ (where C_{AK} is defined as the classification of the apparent key)

This property is an extension of the entity integrity property of the standard relational model designed to deal with security classifications. It means that the apparent key must not have a null value, must be uniformly classified, and its classification must be dominated by the classification of all the other attributes.

MLS integrity property 2: *Null integrity.*

A multilevel relation R satisfies null integrity if, and only if, for each instance of R_c of R, the following two conditions hold:

1. For all $t \in R_c, t[A_i] = \text{null} \Rightarrow t[C_i] = t[C_{AK}]$, i.e., null values are classified at the level of the key.
2. R_c is subsumption free, i.e., it does not contain two distinct tuples such that one subsumes the other. A tuple t subsumes a tuple s, if for every attribute A_i, either $t[A_i, C_i] = s[A_i, C_i]$ or $t[A_i] \neq \text{null}$ and $s[A_i] = \text{null}$.

The null integrity property requires that the null values in a tuple be classified at the level of the key, and that for subjects cleared at higher security levels, the null values visible at lower security levels are replaced by the proper values automatically.

The next property deals with consistency between the different instances of a relation. The inter-instance property was first defined by Denning [28].

MLS integrity property 3: *Inter-instance integrity.*

A multilevel relation R satisfies inter-instance integrity if, and only if, for all $c' \leq c, R_{c'} = \sigma(R_{c'}, c')$, where the filter function σ produces the c'-instance $R_{c'}$ from R_c as follows:

1. For every tuple $t \in R_c$ such that $t[C_{AK}] \leq c'$, there is a tuple $t' \in R_{c'}$, with $t'[AK, C_{AK}] = t[AK, C_{AK}]$ and for $A_i \notin AK$
$$t'[A_i, C_i] = \begin{cases} t[A_i, C_i] & \text{if } t[C_i] \leq c' \\ \langle null, t[C_{AK}] \rangle & \text{otherwise} \end{cases}$$
2. There are no additional tuples in $R_{c'}$ other than those derived by the above rule.
3. The end result $R_{c'}$ is made subsumption free by exhaustive elimination of subsumed tuples.

The inter-instance property is concerned with consistency between relation instances of a multilevel relation R. The filter function σ maps R to different instances of $R_{c'}$ (one for each $c' < c$). By applying the filter function, users are restricted to only that portion of the multilevel relation for which they are cleared.

If c' dominates some security levels in a tuple but not others then, during query processing, the filter function σ replaces all the attribute values that the user is not cleared to see with null-values. A shortcoming in the Jajodia-Sandhu model arising from the use of this filter function was pointed out by Smith and Winslett [75]. Smith and Winslett observed that the filter function σ introduces an additional semantics for nulls. In the Jajodia-Sandhu model, a null value can mean 'information available but hidden' and this null value cannot be distinguished from a null value representing the semantics 'value exists but not known' or a null value with the meaning 'this property will never have a value'.

We stated earlier that the definition of key in the standard relational model is based on functional dependencies (i.e., the value of key attributes functionally determines the values of all the other attributes). Therefore, two tuples with the same values for the primary key attributes must not exist in a relation. Requiring this constraint to hold in multilevel relations may cause indirect release of sensitive information to users with inadequate clearances. The model therefore introduces the notion of apparent primary key, which we described earlier, to distinguish between the apparent key and the *actual* primary key in a relation. The *actual primary key* is defined according to the polyinstantiation integrity property.

MLS integrity property 4: ***Polyinstantiation integrity.***

A multilevel relation R satisfies polyinstantiation integrity if, and only if, for every $R_{c'}$, for all $A_i : AK, C_{AK}, C_i \rightarrow A_i$.

The polyinstantiation integrity property asserts that the apparent key AK specified by a user, its classification, together with the classification of an attribute, functionally determines the value of this attribute. The primary key of a multilevel relation is implicitly defined by this property as the union of the apparent key attributes, their classifications, and the classifications of all the non-key attributes. Formally, the primary key of a multilevel relation is $AK \cup C_{AK} \cup C_R$, where AK is the set of data attributes constituting the user-specified primary key, C_{AK} is the classification attribute for data attributes in AK, and C_R is the set of classification attributes for data attributes not in AK. This is because it follows from polyinstantiation integrity that the functional dependency $AK \cup C_{AK} \cup C_R \rightarrow A_R$ holds, where A_R denotes the set of all attributes that are not in AK. Note that for single-level relations, C_{AK} and C_R will be equal to the same constant value in all tuples. Therefore, in this case, polyinstantiation integrity implies that $AK \rightarrow A_R$, which is precisely the definition of the primary key in relational theory.

2.5.2 Levels of granularity

Security levels can be assigned to data at different levels of granularity. Assigning labels to entire relations can be useful but is generally considered inconvenient. For example, if some salaries are secret but others are not, these salaries must be placed in different relations [27]. Assigning labels to an entire column of a relation is similarly inconvenient in the general case [27]. The finest granularity of labelling is at the level of individual data elements — i.e, element-level labelling. This offers considerable flexibility. Most commercial products offered labelling at the level of a tuple. Although not so flexible as element-level labelling, this approach is considered more convenient than using relation or attribute-level labels

[27]. The following examples illustrate the three labelling schemes used in this thesis. Although the Jajodia-Sandhu model does not specify how the security labels should be stored, the labels are stored as extra columns in the tuple-level and element-level relation instances of our examples as they are specific to relation instances. In the attribute-level example, the labels could be stored either in the relation instance or the schema. Storing the labels in the instance rather than the schema requires significantly more storage space.

1. **For tuple-level labelling**, the Jajodia-Sandhu definition of a multilevel relation represents each tuple t in a relation R without the classification attribute C_i corresponding to each attribute A_i. Instead each tuple t has a corresponding tuple classification attribute TC. The security label for each tuple $t[TC]$ is stored in the relation instance. An example is illustrated in figure 2.3.

DN	**Dept_Name**	**Manpower**	**Dept_Location**	**Manager**	*TC*
D10	Records	43	241 East Annex	Cantor	S
D11	Obstetrics	13	601 Main Building	Baker	TS
D12	Laboratory	8	202 Drew Building	Ebert	S
D13	Geriatrics	12	B16 Basement	Xavier	S

Figure 2.3: Tuple security labelling.

2. **For attribute-level labelling**, the Jajodia-Sandhu definition of a multilevel relation requires that each attribute A_i in a relation R be associated with a corresponding security level C_i, i.e., the tuple classification attribute TC is not required. An example is shown in figure 2.4.

DN	C_1	**Dept_Name**	C_2	**Manpower**	C_3	**Dept_Location**	C_4	**Manager**	C_5
D10	U	Records	C	43	C	241 East Annex	S	Cantor	TS
D11	U	Obstetrics	C	13	C	601 Main Building	S	Baker	TS
D12	U	Laboratory	C	8	C	202 Drew Building	S	Ebert	TS
D13	U	Geriatrics	C	12	C	B16 Basement	S	Xavier	TS

Figure 2.4: Attribute-level labelling.

3. **For element-level labelling**, the Jajodia-Sandhu definition of a multilevel relation requires that each element or attribute value $t[A_i]$ in a relation be associated with a corresponding security level $t[C_i]$. The tuple classification attribute TC is not required. The security label for each element $t[A_i]$ is stored in the relation instance. An example is shown in figure 2.5.

DN	C_1	**Dept_Name**	C_2	**Manpower**	C_3	**Dept_Location**	C_4	**Manager**	C_5
D10	U	Records	TC	43	U	241 East Annex	C	Cantor	U
D11	U	Obstetrics	C	13	C	601 Main Building	U	Baker	C
D12	U	Laboratory	TS	8	U	202 Drew Building	C	Ebert	S
D13	U	Geriatrics	C	12	C	B16 Basement	U	Xavier	TS

Figure 2.5: Element-level labelling.

2.6 Distributed database systems

A *distributed database* as defined by Ozsu and Valduriez [68] is a collection of multiple, logically interrelated, databases distributed over a computer network. A distributed database management system (DDBMS) is also defined in [68] as the software system that permits the management of the distributed database and makes the distribution transparent to users. These two definitions emphasise *logically interrelated* and *distributed over a network* as the property that distinguishes them from other types of systems that have been described as distributed. However, physical distribution does not imply that the computer systems be geographically far apart; they could actually be in the same room. It implies that the communication between them is done over a network instead of through shared memory, with the network as the only shared resource. The DBMS must periodically synchronize the scattered databases to ensure data consistency.

2.6.1 Relation fragmentation

Relations in distributed databases are commonly divided into smaller fragments which are treated as separate relations in their own right. This is commonly done for performance, availability, and reliability reasons. There are three general types of fragmentation alternatives. In one case, called *horizontal fragmentation*, a relation is partitioned into a set of sub-relations each of which is a subset of the tuples (rows) of the original relation. The second alternative is *vertical fragmentation*, where each sub-relation is defined on a subset of the attributes (columns) of the original relation. A less common alternative is the *element fragmentation*, where sub-relation subsets are defined on the elements of the original multilevel relation. The following fragmentation rules taken directly from [68] are required to ensure that the database does not undergo semantic change during fragmentation.

1. **Completeness:** If a relation instance R is decomposed into fragments $R_1, R_2, ..., R_n$, each data item (tuple or attribute) that can be found in R can also be found in one or more of the R_i's. This property, which is identical to the *lossless decomposition* property of relational database normalization, is also important in fragmentation since it ensures that the data in a global relation is mapped into fragments without any loss.

2. **Reconstruction:** If a relation R is decomposed into fragments $R_1, R_2, ..., R_n$, it must always be possible to reconstruct the global relation R from its fragments. This condition is necessary to ensure that global relations can be reconstructed if necessary from the fragments stored in the distributed database. The reconstructability of the relation from its fragments also ensures that constraints defined on the data in the form of dependencies are preserved.

3. **Disjointness:** If a relation R is horizontally decomposed into fragments $R_1, R_2, ..., R_n$ and data item d_i is in R_j, it is not in any other fragment R_k $(k \neq j)$. This criterion ensures that the horizontal fragments are disjoint. If relation R is vertically decomposed, its primary key attributes are typically repeated in all its fragments. Therefore, in the case of vertical partitioning, disjointness is defined only on the nonprimary key attributes of a relation.

Our proposed kernelized approach to multilevel database security draws on these fragmentation techniques for fragmenting and distributing multilevel relations based on security level. A distributed database is logically similar to a multilevel database composed of relations which are fragmented and distributed on the basis of security level. If the granularity of the security levels is at the tuple-level, the single-level fragments are logically the same as a horizontally distributed database. Attribute-level granularity results in vertically distributed databases. If the granularity is based on element-level labels, the distribution represents neither a horizontally distributed database nor a vertically distributed database.

2.6.2 Query processing

The retrieval function of the data manipulation language (DML) is typically executed more often than the update function and has become a specialised function referred to as query processing. Questions are directed at a database using high level query languages such as SQL. A disadvantage of using high level languages to write queries is that it is possible to write queries that take a very long time to execute. Because of this, much research has been conducted into query optimisation techniques. Good general treatments are included in most books discussing the relational model [24, 35]. A more advanced treatment of query processing may be found in [54].

The fragmentation of database relations introduces a new problem, that of handling user queries that were specified on entire relations but now have to be performed on sub-relations. Specifically, the issue is one of finding a *query processing* strategy based on the fragments rather than the relations, even though the queries are specified on the latter. Typically, this requires a translation from what is called a *global query* to several *fragment queries.*

The studies of query processing that have had the most influence on this thesis are those involving distributed query processing [16, 18, 19, 68]. These sources contain detailed treatments of the process of converting global queries into their horizontal, vertical and mixed fragment equivalents. These query fragments are processed at the appropriate databases. While the process was developed for reasons other than security (for example, economies of scale), the theory behind query processing in distributed databases is useful in ensuring security in multilevel databases. This thesis builds on this recognition of the similarities in the two processes for the treatment of single-level fragments.

2.7 Summary

A database system is a computerised system to keep track of information. A relational database model represents a database as a set of tables. This simple idea makes it easy to understand the model. More importantly, the relational model is directly related to the mathematical concept of a relation. This mathematical basis has resulted in the definition of languages (relational algebra and relational calculus) which can be used to express the operations that can be performed on the tables.

While the classical relational model is concerned with data without security classifications, the concern of the multilevel relational model is data with security classifications. The security labels in a multilevel relational model are drawn from a set of partially ordered elements. Relations may be single level or multilevel. In single-level relations, all of the attributes have the same security classification. In multilevel relations, the attributes have different security levels.

The BLP model defines the read and write rules that must hold in an MLS system in order for security to be preserved. These rules are based on the simple security property and the *-property. The simple property states that in order to 'read' an object, the subject's security level must dominate the object's security level. The *-property states that in order to 'write' an object, the object's security level must dominate the subject's security level.

A distributed database is defined as a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (DDBMS) is also defined as the software system that permits the management of the distributed database and makes the distribution transparent to users. Of particular interest to our research is the fragmentation and query processing features of a DDBMS.

Chapter 3 will present a review of related research and developments in MLS/DBMS with an emphasis on the models, prototypes and products that have had the most impact on our research.

Chapter 3

Related Research and Developments

3.1 Introduction

Research on multilevel security models began in earnest in the early 1970s with the aim of producing high-level, software-independent, conceptual models that describe the protection mechanisms of a system [15, 58]. In order to have assurance that a given model would perform as specified, researchers also recognised the need to formalise these models. Many of the models were based on the military model of security, but there were some notable studies of security policy in the commercial sector [13, 15]. The SeaView model was the first formal MLS model. Efforts at implementing MLS systems began in the early 1980s; these efforts were either research-driven, resulting in a prototype, or commercially-driven, resulting in a commercial product. Prototypes developed in the former category include SeaView, Lock Data Views, A1 Secure DBMS, and SWORD. Products that resulted from efforts in the latter category include Trusted Oracle 7, Sybase Secure SQL Server 11.6, INFORMIX-OnLine/Secure DBMS 5.0, and Open INGRES/Enhanced Security 1.2. Although it is widely accepted that implementing a MLS/DBMS is a highly complex task, only one study by Thuraisingham and Kamon [77] examines the impact of this complexity on performance.

The next section will briefly review some of the MLS models that have had some influence on this thesis. Section 3.3 considers the architectures for multilevel database systems that resulted from the Multilevel Data Management Security Summer Study (or Woods Hole study group) [1]; these include the integrity lock, kernelized, replicated, and trusted subject architectures. Section 3.4 presents a number of MLS/DBMS prototypes that have been developed over the last two decades, mainly in response to the proposals of the Woods Hole study group. A survey of commercial MLS/DBMS products that were developed to address security concerns in different application areas is presented in section 3.5. Section 3.6 describes one previous effort at building a distributed secure system from a collection of COTS components, and another effort which proposed using a hardware device to filter communication between System-high and System-low LANs. A brief review of some database benchmarking suites and a database performance study are presented in section 3.7. The chapter is summarized in section 3.8.

3.2 Overview of MAC models

The *SeaView model* [28] was the first formal MLS relational model. It was subsequently developed as a prototype of an MLS/DBMS based on the view mechanism. In the SeaView model, data elements are classified at the element-level and stored as single-level fragments. The single-level fragments are created from multilevel relations which are partitioned based on security level. Each of the single-level fragments contains the primary key of its parent relation. When a user requests information, a view is created consisting of those single-level fragments that the user is authorised to access. The *Jajodia-Sandhu model* [46, 48, 50] is based on the SeaView model and addresses some of the flaws identified in the SeaView model [46, 48, 50] such as the proliferation of tuples during updates, and spurious joins. The *Commutative Filter Model* [26], advocates replacing the reference monitor that enforces mandatory access control with a trusted filter inserted between the user and the DBMS which would intercept returning query answers from the database and remove all data the user is not entitled to see; it is derived from the maximal authorized view approach proposed by Downs and Popek [32].

The Keefe et al. query modification approach [52] modifies a query by searching a security constraints database for all rules involving the query in question. The restrictions in these rules are then added to the original query in order to make it safe. Data objects in the database are assigned security labels by the model based on its own security constraints. The *Smith-Winslett model* [15, 75] addresses the semantics of an MLS database based on the concept of belief, where a user sees and believes the contents of the database at its own level, and sees the data objects at lower security levels. For example, a Secret user can see Secret, Confidential, and Unclassified objects, but it believes only that Secret objects accurately represent the real world situation, and it makes no assumptions about Confidential and Unclassified objects. The Smith-Winslett model is also known as *belief-based* semantics. Unlike the Jajodia-Sandhu model, this model does not support classification at the level of each single attribute. Instead, access classes can be assigned only to key attributes and to tuples as a whole. The most recent of the MLS data models, the *Multilevel Relational model* (MLR) [20, 50] was proposed by Chen and Sandhu and is substantially based on the Jajodia-Sandhu model. It combines ideas from SeaView, belief-based semantics, and the Lock Data Views model in trying to eliminate ambiguity whilst retaining upward information flow. This model supports labelling at the element-level of granularity. A detailed description of these models may be found in [15, 38, 58].

3.3 Architectures for multilevel database systems

Mandatory access control is often implemented by classifying both users and data at various security levels; however, it can also be implemented by classifying the sensitive data only at *system high.* This approach requires all users to be cleared at system high. The major advantage with this approach is that existing DBMSs can be used with no change; however, it suffers from certain drawbacks. First, it is very expensive since the cost of clearance is high. Second, it involves additional security risk since more people are given the highest clearance. In order to try and resolve this problem, the Woods Hole study group, organized by the U.S. Air Force in 1982 [1, 6], considered different architectures for building MLS/DBMSs. The study group proposed requirements for a purpose-built multilevel DBMS as well as solutions that looked at how existing DBMS technology could be reused to build MLS/DBMSs [33]. Among the many architectures proposed by this study for the physical storage of multilevel data, three solutions became prominent; these include: the *Integrity lock (or spray paint) architecture*, the *Kernelized architecture*, and the *Replicated architecture* [1].

3.3.1 The integrity lock (or spray paint) architecture

This architecture uses a single DBMS to manage all levels of data [39]. Figure 3.1 is depicted with the four security levels introduced in our earlier example of a simple lattice in figure 2.2(A). The trusted front end (TFE) (or trusted filter) acting as the TCB is responsible for enforcing multilevel protection; it does this by attaching security labels to database objects in the form of a cryptographically generated stamp (or paint) which it then assigns to every data item. While retrieving data, the TFE checks the stamp on the data item and makes sure that the simple-security property of the BLP policy is met. Since a user's query has access to the entire database, a clever user can infer higher level information by formulating a query that modulates its output based on high level data. For example, consider the relation in figure 3.2, and suppose a Secret user submits the following query:

```
If(Manpower > 50) return 1, else return 0
```

The value that is returned to the Secret user in response to this query will not have a stamp that is higher than Secret, but information can still be inferred from the query. This architecture can thus be exploited to open up high bandwidth covert channels.

3.3.2 The kernelized architecture

In this architecture, illustrated in figure 3.3, the multilevel database is partitioned into single level databases, which are then stored separately [1, 42, 60]. In this architecture,

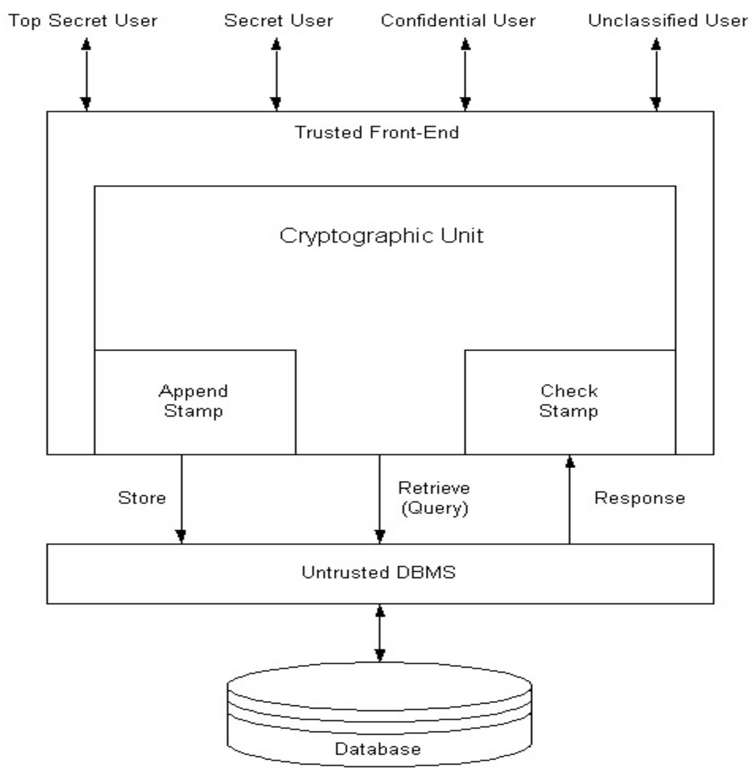

Figure 3.1: The integrity lock (or spray paint) architecture.

DN	Dept_Name	Manpower	Dept_Location	Manager	Stamp
D10	Records	40	241 East Annex	Cantor	S
D11	Obstetrics	83	601 Main Building	Baker	TS
D12	Laboratory	45	202 Drew Building	Ebert	S
D13	Geriatrics	32	B16 Basement	Xavier	S

Figure 3.2: Vulnerability of the integrity lock architecture.

there is a separate DBMS for each security classification. The trusted front-end ensures that the user's queries are submitted to the DBMS with the same security level as that of the user, while the trusted back-end makes sure that a DBMS at a specific security level accesses data without violating the mandatory security policies. Processing a user's query that is requesting data from multiple security levels involves expensive Joins that may degrade performance, since the different levels of data are stored separately. On the other hand, since this architecture has separate DBMSs for each security level, the scheduler that is responsible for concurrency control can also be separated for each level as it need not be part of the TCB [1, 60]. Its concurrency control technique has to address two basic challenges: security and data availability. When a high level process wants to read a low level data item, it cannot set a read lock on the data item since this process would introduce a covert channel – the locking and unlocking of low level data can be observed by a low level process. Trying to eliminate this problem causes the high user to wait for the low level data, which creates a problem of data availability. A solution suggested in [1, 60] is to use multiversioning, hence allowing high users to read older versions of low data. Other researchers [1, 60] have also suggested that the number of versions be restricted to two, one for the subjects at the same level of data item and one for subjects at higher levels.

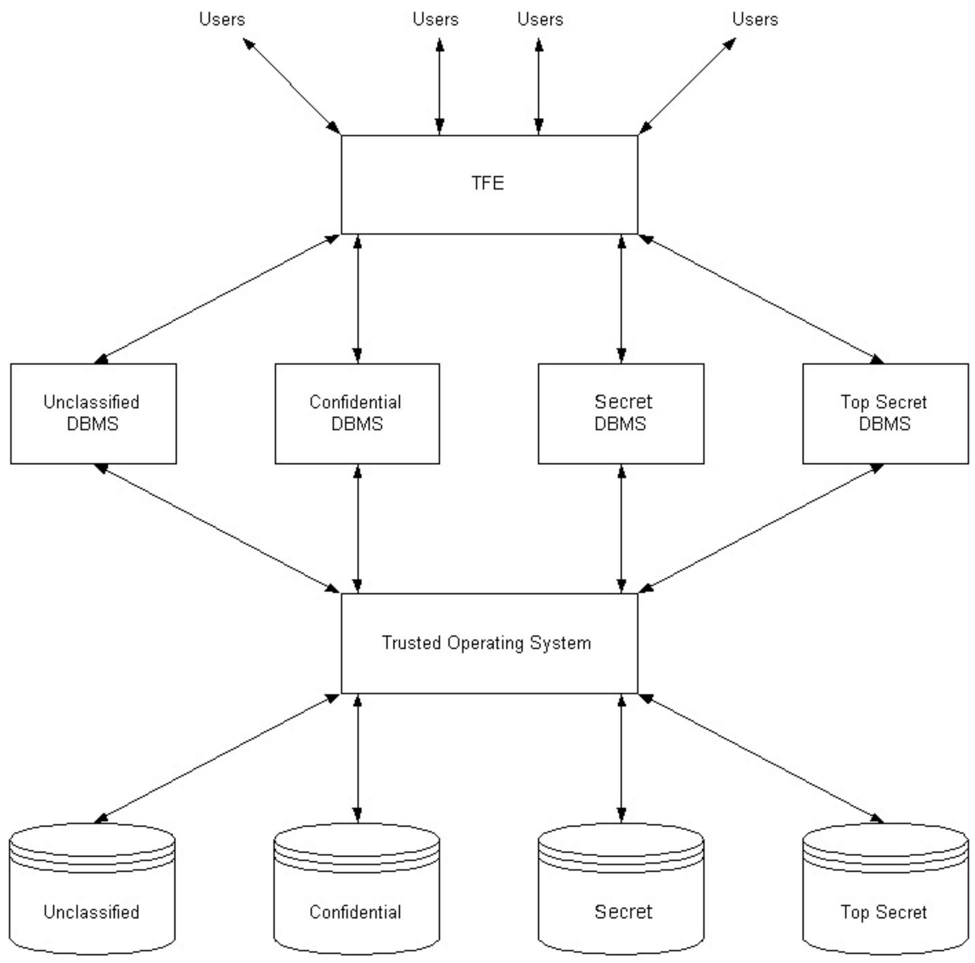

Figure 3.3: The kernelized architecture.

3.3.3 The replicated architecture

This architecture, illustrated in figure 3.4, uses a separate database management system to manage data at or below each security level [22]; a database at a security classification contains all information at its class and below; lower level data are therefore replicated in all databases containing higher level data. The replicated architecture suffers from the following two difficulties, arising since lower level data is replicated in higher level databases. First, propagating updates of the lower level data to higher level DBMSs in a secure and correct manner is not simple. Second, this architecture is impractical when there are a large number of security levels.

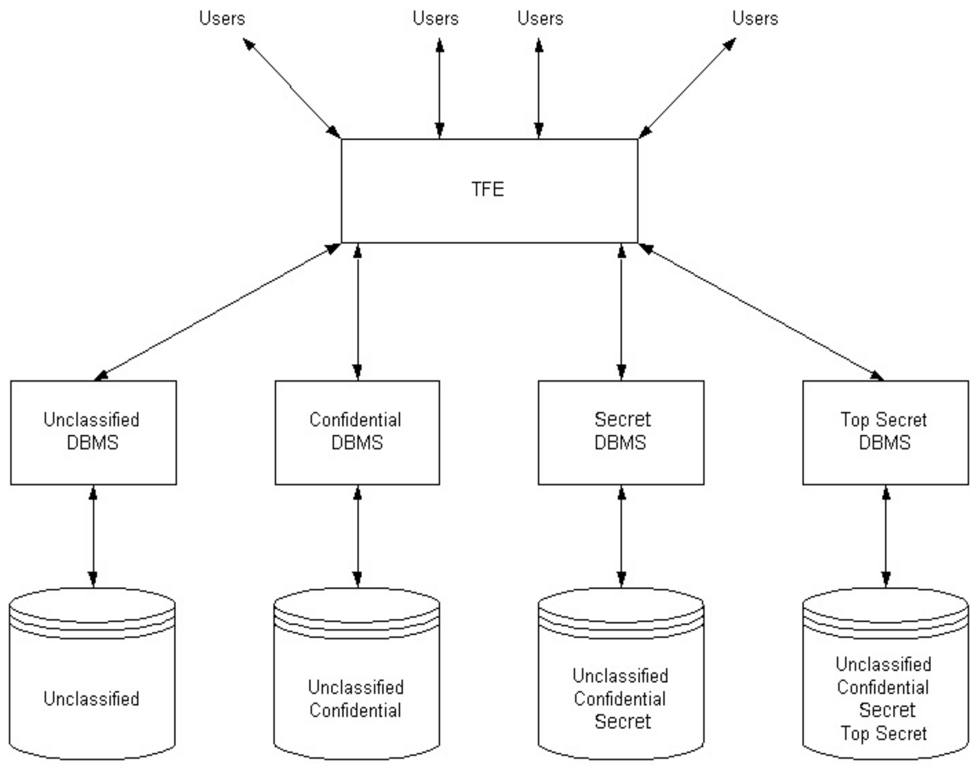
Users
Users
Users
Users
TFE
Unclassified DBMS
Confidential DBMS
Secret DBMS
Top Secret DBMS
Unclassified
Unclassified Confidential
Unclassified Confidential Secret
Unclassified Confidential Secret Top Secret

Figure 3.4: The replicated architecture.

3.3.4 The trusted subject architecture

The long-term solution proposed by the summer study group was to build a completely trusted DBMS from scratch; this is referred to as the *Trusted subject architecture* and is illustrated in figure 3.5. In this architecture, all DBMS data is stored with the DBMS label, and only DBMS code can be executed by a subject with the DBMS label. Subjects with a DBMS label are trusted and are thus exempt from mandatory access control restrictions. Since the DBMS is trusted, a trusted front-end is not required in this architecture. Instead, users communicate with the DBMS through an *Untrusted Front End (UFE)* at their respective classification level.

3.4 Multilevel secure prototypes

In response to the Woods Hole summer group proposals, a number of development efforts were initiated; some of these efforts resulted in the deployment of MLS/DBMS prototypes or products. This section describes the prototypes resulting from some of these efforts.

Trusted systems, especially DBMSs, tend to be quite large with respect to the

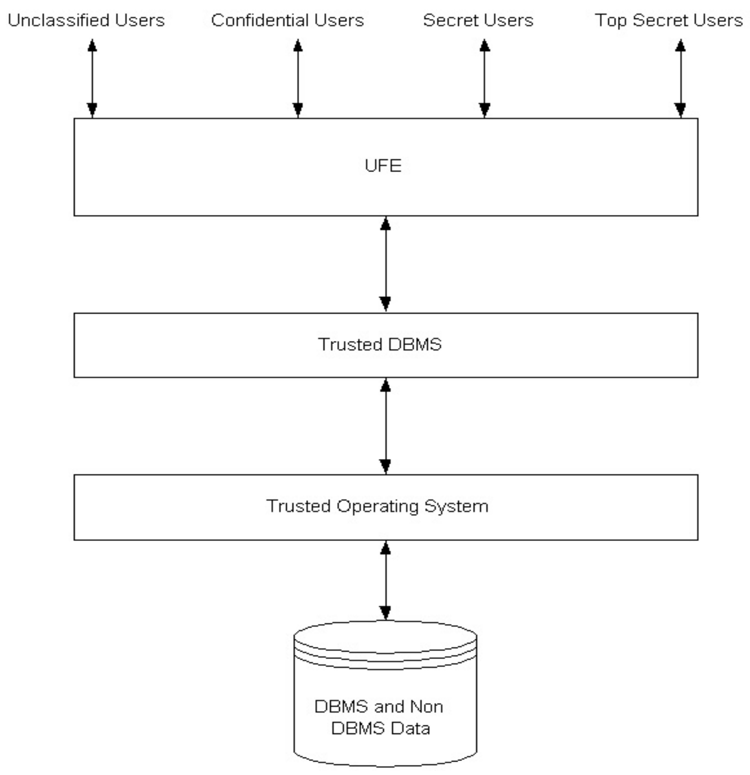

Figure 3.5: The trusted subject architecture.

amount of code needed for their implementation. Although we stated the desirability of high assurance for trusted systems in section 3.3.4, with today's technology a complete formal proof of the specifications of such large systems is not yet possible; a great deal of research on formal specification and verification is ongoing. The enormous amount of code necessary for implementing trusted DBMSs was largely responsible for the conservative approach taken by most trusted DBMS developers in exploiting an approach known as *TCB subsetting*; the use of this approach is recommended for achieving higher levels of assurance. The approach requires trusted DBMS developers to reuse and build on previously built and verified trusted systems. TCB subsetting was identified as a strategy for building trusted DBMSs in the Trusted Database Management System Interpretation (TDI) [66]. This section discusses the most prominent projects undertaken over the past ten years which have had the goal of building systems that meet the requirements of the higher levels of trust specified in the TDI evaluation criteria.

We have identified four major efforts over this period to design and implement trusted relational database prototypes. The first effort was a prototype of the SeaView Trusted DBMS implemented by Stanford Research Institute (SRI) International, Gemini Computers, and Oracle Corporation. The second effort was an implementation of the Lock Data Views (LDV) approach by Honeywell. The third implementation, called the A1 Secure DBMS, originated from an internal TRW research and development project. The

SWORD Multilevel Secure DBMS was the fourth effort, developed by the UK's Defence Research Agency.

3.4.1 SeaView prototype

The most ambitious proposal for the development of a trusted DBMS came from the SeaView project [60, 61]. The goal of this project was to design a prototype MLS/DBMS based on the SeaView model.

The most significant contribution of SeaView was the realisation that multilevel relations need only exist at a logical level and can be decomposed into single-level base relations. The advantage of this observation was mostly practical. In particular, single level base relations can be stored using conventional DBMSs and commercially available TCBs can be used to enforce mandatory controls with respect to single level fragments.

The architectural approach taken by the SeaView project was to implement the entire DBMS on top of the commercially available Gemsos TCB, which uses a dedicated hardware platform [74]. Gemsos provided user identification and authentication, maintenance of tables containing clearances, as well as a trusted interface for privileged security administrators. Multilevel relations were implemented as views over single-level base relations. The single-level relations are transparent to users; they are stored using the storage manager of an Oracle DBMS engine. From the viewpoint of Gemsos, every single-level relation is a Gemsos object of a certain access class. Gemsos enforces a mandatory access security policy based on the BLP security paradigm. A label comparison is performed whenever a subject attempts to bring a storage object into its address space. A subject is prevented from accessing storage objects not in the subject's current address space by means of hardware controls included in Gemsos.

In addition to mandatory access controls, the SeaView security policy requires that no user is given access to data unless that user has also been granted discretionary authorisation to the data. DAC protection is performed outside Gemsos and allows users to specify which users and groups are authorised for specific modes of access to particular database objects, as well as which users and groups are explicitly denied authorisation to particular database objects.

Since a multilevel relation is stored as a set of single-level fragments, two algorithms are necessary:

- A *decomposition algorithm* which breaks multilevel relations into single-level fragments.
- A *recovery algorithm* to reconstruct the original multilevel relation from the fragments. It is obvious that the recovery process must be lossless (i.e., it must recover the multilevel relation in its entirety), otherwise recovery is incorrect.

In SeaView, the decomposition of multilevel relations into single-level ones is performed by applying vertical and horizontal fragmentation; recovery is performed by Union and Join operations. As an example, consider a conceptual multilevel relation R with the schema: $R(A_1, C_1, \ldots, A_n, C_n, TC)$, where each A_i is an attribute defined over a domain D_i, and each C_i is a security classification drawn from the set: $[U, C, S, TS]$, where $U \leq C \leq S \leq TS$. We assume that A_1 is the apparent primary key. The original SeaView decomposition algorithm consists of the following three steps:

- **Step 1:** The multilevel relation R is vertically partitioned into n projections $R_1[A_1, C_1]$, $R_2[A_1, C_1, A_2, C_2]$, $\ldots$, $R_n[A_1, C_1, A_n, C_n]$.
- **Step 2:** Each R_i is horizontally fragmented into one resulting relation R_{ij} $(1 \leq j \leq 4)$ for each security level. Obviously, for the set of classification levels $[U, C, S, TS]$, this results in $4n$ relations.
- **Step 3:** In a further horizontal fragmentation, the relations R_{ij}, $2 \leq i \leq n$, (i.e., $4n - 4$ relations) are further decomposed into at most 4 resulting relations. This final decomposition is necessary because of SeaView's support for polyinstantiation.

A performance analysis of this algorithm by Jajodia and Sandhu [46] pointed out that the decomposition algorithm leads to unnecessary single-level fragments, and that performing the recovery of multilevel relations requires repeating joins that may lead to spurious tuples. The performance of SeaView is thus highly dependent on the efficiency of these decomposition algorithms.

3.4.2 Lock Data Views prototype

Lock Data Views (LDV) [42] is another MLS/DBMS prototype that was developed during the same period as SeaView. It was hosted on the Lock TCB and prototyped at the Honeywell Secure Computing Technology Center (SCTC) and MITRE. It supported a discretionary and a mandatory security policy. One aspect of this prototype that was of special interest for the increased functionality of its TCB is *type enforcement*.

The general concept of type enforcement in Lock and its use in LDV is discussed in [42]. The main idea is that the accesses of a subject to an object are restricted by the role the subject is performing in the system. This is done by assigning a domain attribute to each subject and a type attribute to each object, both maintained within the TCB. Entries in the domain definition table correspond to a domain of a subject and to a type list representing the set of access privileges the subject has within the domain. Lock's type enforcement mechanism made it possible to encapsulate LDV in a protected subsystem, by declaring the database objects to be special Lock types (Lock files) which are only accessible to subjects executing in the DBMS domain. Since only DBMS programs are allowed to execute in this domain, only DBMS processes can access the Lock types holding portions of the database. A solution to the problem of secure release of data from the DBMS domain to the user domain was made possible through Lock's support for the implementation of *assured pipelines*.

Two basic extensions to the Lock security policy were implemented in LDV [43]. Both extensions concern the proper classification of data. The first extension deals with the insert and update of data. During insert and update, the data is assigned to the Lock type classified with the lowest level at which the tuple can be stored securely. The second extension is concerned with query results. The result of a query is transferred from Lock types into ordinary objects and the appropriate security level of the query result is derived. The two policies are enforced in LDV by using three assured pipelines. These three pipelines are the query/response pipeline, the data/input pipeline, and the database definition/metadata pipeline.

The *query/response* pipeline is the query processor of LDV. The pipeline consists of a set of processes which execute multi-user retrieval requests, integrate data from different Lock types, and output the information at an appropriate security level. First, a user-supplied query is mapped from the application domain to the DBMS domain, then the query is processed, the result is labelled and finally it is exported to the user. In order to mitigate against the possibility of logical inference over time, the response pipeline included a history function. This mechanism was designed to trace the queries already performed for a particular user and to deny access to relations based on the query history of the user.

The *data/input* pipeline is responsible for actions that need to be taken whenever a user issues an insert, modify, or delete request. First, the request has to be mapped from the application domain to the DBMS domain where the request can then be processed. LDV does not support blind writes; for example, a delete request will only affect data at a single classification level. For consistency reasons, data is not actually removed but marked as deleted. Before the actual removal takes place, certain consistency checks are performed. More complicated is the case where the request is an insert operation. Classification rules that may be present in the data dictionary (see database definition/metadata pipeline) may make it necessary that the relation tuple is decomposed into different subtuples that are stored in separate files, each with a different classification. The modify request operates in a similar way to the insert operation.

The *database definition/metadata* pipeline interacts with the LDV data dictionary and was used to create, delete, and maintain metadata. Metadata may either correspond to definitions of the database structure (relations, views, attributes, domains) or the classification constraints. Classification constraints are rules that are responsible for assigning proper classification levels to the data. The use of the metadata pipeline is restricted to a database administrator or database security officer (DBSO). Here, again, Lock type enforcement mechanisms were used to isolate the metadata in files that can only be accessed by the DBMS domain and the DBSO domain, and not by the application domain.

Furthermore, data is distributed across Lock files by assigning a set of files per security level. The data/input pipeline determines the appropriate assignment of data to files by examining the classification constraints stored in the data dictionary. In LDV, there is no replication of data across different security levels. The advantage of this approach is the simplicity of updates. However, there is an inherent and significant performance penalty for retrieval requests arising from the necessity for a recovery algorithm in LDV; this is described in [42].

3.4.3 A1 Secure DBMS prototype

The A1 Secure DBMS (ASD) [83], implemented on top of an existing DBMS called ASD, was an MLS/DBMS prototype developed by a research project at TRW. ASD can be operated in three different modes. Under the first mode of operation, ASD can function as a DBMS server on a local area network. Under the second mode of operation, ASD can serve as a back-end DBMS for various single-level or multilevel host computers. Under the third mode of operation, ASD can serve as a resident DBMS within a multilevel host running a secure operating system.

The mandatory object of protection in ASD is the tuple of a table. The mandatory security policy enforced satisfies the BLP security policy. Tuples also inherit their discretionary access from the tables in which they are located. Discretionary access is specified for tables in terms of permissions for access and denials of access. Permissions and denials may be specified with respect to individual users, groups of users, or ‘public’. The permissions are Select, Insert, Delete, and Update. The most specific discretionary access specification takes precedence over a less specific discretionary access specification and a denial (at a given specificity) takes precedence over a permission. For example, a user is more specific than a group which is more specific than public.

ASD also enforces the Biba integrity model [10] which states that a subject may read a tuple if, and only if, the integrity level of the tuple dominates the integrity level of the subject. A subject may write a tuple if, and only if, the integrity level of the subject dominates the integrity level of the tuple.

The ASD system code is divided into two groups: trusted code and untrusted code. Earlier, in subsection 3.3.4, we stated that a trusted subject is one which is exempt from mandatory access control; conversely, an untrusted subject is one which is not exempt from mandatory access control. The trusted code in ASD is part of its TCB. The untrusted code is replicated by security level into separate, untrusted processes. The TCB ensures that each untrusted process can send and receive data only at the security level of the process. Each untrusted process supports a single user. The security level of the user’s untrusted process is the same as the user's session. If the user is accessing ASD while operating on a multilevel secure operating system, then the level of the ASD process is the same as the level of the user's process on the multilevel operating system. If the user is accessing ASD from a single security level host or workstation, then the security level of the ASD process is the same as the security level of the user's workstation or host. The secure operating system provides the following services to the DBMS:

- separation of trusted processes from untrusted processes and untrusted processes from each other;
- secure communication services between untrusted processes and the DBMS kernel via message passing to the secure operating system;
- protection of the file holding the database data, such that only the DBMS kernel can have direct access to it;
- user authentication;
- trusted path.

The file containing the database data is protected by the mandatory access policy of the secure operating system. The database file is labelled at System-high, the same level at which the DBMS kernel is running. The secure operating system also enforces discretionary access control on the DBMS data file. The DBMS data file can only be accessed by the special user "DBMS". In addition, the DBMS file is assigned a special integrity compartment, to prevent any process other than the trusted DBMS kernel from modifying the data in the file, including the security labels of the rows.

In operation, a query is formulated in the host (or application process if ASD is used in a stand-alone mode) and sent to the ASD server. The trusted interface ensures that the request is serviced by the appropriately classified untrusted DBMS code. This code processes the request and sends requests to trusted DBMS reference monitor code to actually retrieve the data. Various trusted utilities are present in the system to create and maintain the ASD database.

ASD has the capability for running multiple instantiations of the untrusted DBMS code, each at the same level as the host application process that it is supporting. This process is only given the data it is permitted to see according to the ASD security policy. It is only given access to data whose security level it dominates. It can only write objects at the same level as the process in which it is currently executing. The security levels of newly created tuples are equal to the security level of the untrusted DBMS process that requested the tuple creation.

Since ASD will operate under the control of a secure operating system, some security functions such as identification and authentication, that are normally associated with a secure system, are not part of ASD, but are provided by the secure operating system.

3.4.4 The SWORD prototype

Driven by concerns about the adverse effects of polyinstantiation on integrity constraints, SWORD [85] was developed as an MLS/DBMS at the Defence Research Agency of the United Kingdom to offer an alternative approach to confidentiality control. SWORD enforces security at the element-level of granularity. Previous approaches to protecting the confidentiality of information have relied mainly on classifying views or polyinstantiation. The abstract idea that is the basis of SWORD is referred to as the *Insert Low Approach*. This approach allows the DBMS to check whether integrity constraints are upheld without compromising the confidentiality of information.

A major design consideration for SWORD was the threat of covert channels that could emerge from information flows. The insert low approach controls the insertion and deletion of records in a table in such a way that a user may only insert into or delete from a table if no other user exists with a lower or incomparable clearance that can access the table. For example, a user with a clearance of Unclassified is always permitted to insert and delete rows because any user with a higher clearance, say Top Secret, that receives information as a result of these insert or delete operations must be cleared to handle it anyway because upwards information flow is permitted. The insert low technique is further generalised in SWORD by designating a different 'bottom' classification to each table: the *table classification*.

The concept of *place holders* also emerged from the SWORD DBMS effort. Place holders are described in [85] as fields inserted into a table by a user cleared at level System-low. These fields have a classification that dominates the user's clearance (i.e., they are 'overclassified' relative to the user's clearance). From a confidentiality point of view, the data within the place holder is only overclassified and is not considered important. Place holders are used in the first part of what is a two stage operation to insert a record. In the first stage of the operation, a user cleared at level System-low inserts a record into the table, comprised of data elements with different security classifications. Any field that is designated for data elements classified above the Unclassified level is given a place holder value. In the second stage of the operation, the place holder is replaced with the relevant data that was really intended for that field. This update query is performed by a user cleared above the Unclassified level, say a Secret user; this user is not permitted to change the classification of the field. Unclassified users cannot detect this update as they are prevented from observing the contents of Secret fields.

SWORD supports the basic operations of Insert, Select, Update, and Delete. The

SWORD DBMS interface is accessible through Secure SQL (SSQL), essentially a standard SQL extension that can handle classifications. The insertion of rows is accomplished using place holders; the fields of a row may contain any classification level that dominates the clearance of the user but, without further constraints, this could open the possibility for a denial of service problem. As an example of this denial of service problem, a user could insert a place holder field in a record and assign it a Top Secret classification. Although the field value is unimportant, users whose security clearance is greater than Unclassified and less than Top Secret, e.g. Secret users, can no longer get complete answers to their queries.

A retrieval operation is permitted only if a user's clearance dominates the classification of the table and, more specifically, only if the clearance of the user dominates the field's classification. The data in a field is replaced by a special value if a user cannot examine it; this value will indicate that they are not cleared to see it. However, the classification of a field is not hidden as all classifications are assigned by Unclassified users. In order to update a field, the clearance of the user must be dominated by the classification of the field; this implies that users can update fields that they cannot observe. Rows in a table can be deleted by a user if, and only if, the clearance of the user either equals the classification of the table or is at System-low.

3.4.5 Purple Penelope

Purple Penelope [84] was developed as a prototype secure system at the Defence Research Agency of the United Kingdom to show that the security functionality of Windows NT[1] can be extended to provide labelling, and other security mechanisms, to support users who must handle sensitive information. The architecture provides each user with a private desktop in which to work, along with services for sharing data. Within a desktop, the user is helped to attach security labels to his or her data. When data is shared, labelling prevents accidental disclosure, but other measures defend against other forms of compromise. Although Windows NT does not provide any direct support for labelling functionality, the prototype customises Windows NT to provide discretionary labelling, easy to use role-based access controls, and effective accounting and audit measures for shared files.

The functionality provided by the system is intended for use within domains that work in System-high or compartmented mode — it was not designed to be appropriate for multilevel mode. The assured discretionary labelling functionality prevents those users with inadequate clearance from observing some data directly, but users with adequate clearance may use their discretion to relabel data, typically by copying it and giving the copy a lower label than the original. Tighter controls are imposed on the exchange of data between domains. In particular, assured controls prevent data being exported from

[1]Windows NT is a product of Microsoft Corporation.

a domain without this being sanctioned by one of the domain's members using a trusted path. Accounting and audit functionality is used to monitor what data is exported in order to detect, at a later date, inappropriate behaviour amongst users. Firewalls are deployed to prevent, in a proactive way, any inter-domain communication that is not required.

The Windows NT operating system, and the Microsoft applications that run on it, contain many open extensible interfaces which allow them to be customised with third party value-added services. The 'Purple Penelope' prototype has exploited these interfaces to extend Windows NT's security functionality. The author points out that the additional security functionality provided by Purple Penelope was implemented with a modest amount of code, and works in systems with a heterogeneous mixture of Windows NT and Unix servers, including Secure Unix servers such as compartmented workstations.

Windows NT already provides assured security functionality, but this only allows access to data to be controlled according to user identity rather than by security label. Purple Penelope exploits the native security functionality of Windows NT as the basis for an implementation of discretionary labelling whose assurance is largely derived from that of the base product. The visible manifestation of the additional security is the appearance of a stripe across the top of the screen. This displays the security marking of the application or data that currently has focus. Depending upon the application, the marking may be associated with an entire application, an individual document or a database field. The screen stripe also displays the marking associated with the data in the clipboard. The marking of other data which is visible on the screen, but which does not have focus, is not displayed in the stripe, although some applications may display markings in their window alongside their data.

Purple Penelope provides each user with a private filestore and access to a shared filestore. Applications are free to read and write files in the user's private filestore, but they cannot access the private filestore of any other user. Applications can read files in the shared filestore if the user has sufficient clearance and role-based access rights, but applications cannot modify any shared files. Applications may export files to the shared filestore by copying them, removing files from the shared store, or re-labelling shared files; however, these are accountable actions which must be sanctioned by the user and are subject to role-based (access) controls. The software which solicits the user's approval establishes a trusted path and cannot be bypassed by the applications.

The marking applied to selected data may be changed using a dialogue box activated by clicking on the screen stripe. For data which is private to the user, the action is not audited, even if a lower marking is applied. For shared files, however, the action is subject to role-based controls and is noted so the users can later be held to account for their actions. Users can exchange messages, which may have attached files. Independent labels are applied to the message body and any attachments. Checks are made to ensure messages cannot be sent to users with inadequate clearances. Users may access services hosted on Unix servers. Services provided by the prototype include discretionary labelling, role-based controls, track management system, export sanction control, and accounting and audit functionality.

3.5 Commercial MLS/DBMS products

Besides the research prototypes described in the previous section, during the same period, several vendors also released commercial systems that supported mandatory access controls. This section surveys a number of commercial MLS database products and their functionalities. These products provided security within the database application itself. The products include Ingres, Oracle, Informix, and Sybase. Three of these products are no longer being supported by their respective vendors; the fourth, Oracle, only receives partial support, provided in the form of a supplementary 'labelled security' add-on feature for the standard distribution of Oracle DBMSs. This feature may be added to the standard Oracle DBMS distribution to provide additional security, but is not supplied as a separate autonomous application.

Open INGRES/Enhanced Security 1.2 was developed, distributed, and supported by Computer Associates until August 2000 as a fully featured multilevel relational DBMS offering an ANSI compliant SQL interface. In addition to the standard discretionary access controls, it provided security auditing and mandatory access control features. INGRES/Enhanced Security acted as the primary component in the security of the system by providing a set of database security functions that covered the areas of identification, DAC, MAC, accountability, audit, and object reuse. The product provided support for a variety of decision support and application tools including the OpenINGRES/Replicator, Open Road products (such as Vision and Windows4GL) as well as various third generation languages.

Trusted Oracle 7 was supplied by Oracle Corporation as a multilevel relational DBMS, providing all the features of Oracle 7, with the added functionality of mandatory access control and labelling. Trusted Oracle 7, release 7.2, included all the security functionality of Oracle 7 release 7.2, including granular privileges for enforcement of least privilege, user-configurable roles for privilege management, flexible auditing, stored procedures and triggers for enhanced access control and alert processing, row-level locking, robust replications and recovery mechanisms, secure distributed database communications and the ability to use external authentication mechanisms. In addition, Trusted Oracle 7 provided a full set of multilevel security functionality including flexible label management and policy enforcement, multiple security architectures, and information flow and dissemination control. The SeaView prototype and Trusted Oracle 7 are both based on the kernelized architecture approach. Although Oracle continues to provide a 'labelled security' extension for its standard DBMS, its support for Trusted Oracle came to an end in 1998.

INFORMIX-OnLine/Secure DBMS 5.0 was distributed by Informix Software Ltd as a relational DBMS for multilevel, multi-user, multi-tasking UNIX platforms and is based on the commercial INFORMIX-OnLine database server. The DAC policy is implemented by permitting owners of DBMS objects to grant various privileges on those objects, allowing them to be shared with other users. It is completely separate from the DAC policy enforced by the operating system. Resources subject to the DAC policy include the DBMS databases and tables. Appropriate database privileges must be held by users who want to access tables within a specific database, in addition to privileges to the specific target table. In addition to the DAC mechanisms, OnLine/Secure extends the system-wide MAC policy to DBMS objects. This extension enables the assignment of labels to databases, tables and rows. The MAC mechanism provides control over the distribution of data protected by the system to only those users with appropriate authorisations, and enforce a policy which is fully consistent with the BLP security policy. OnLine/Secure relies on the operating system administrator to define the labels that are to be used in the enforcement of the system-wide policy. Operating system services are used to aid in the enforcement of the security policy when accessing DBMS objects. Informix Software ceased supporting this product in October 1999.

Sybase Secure SQL Server 11.6 was a security enhanced version of the Sybase SQL Server running on an HP-UX operating system platform. It was distributed by Sybase Inc. until 2000 and included security mechanisms for identification and authentication, MAC, DAC, configurable auditing, groups, and roles, as well as integrity features such as triggers, stored procedures, and declarative and procedural referential integrity. Its identification and authentication mechanism is separate from that of its HP-UX operating system platform. The Secure SQL Server provides flexible DAC through the SQL grant and revoke mechanism. Groups, roles, and individuals may have access granted or revoked to databases, tables, views, stored procedures, and columns. Subsequent grants and revokes can affect all or part of the privileges previously granted or revoked, providing the ability to define complex access control policies with a series of simple grant and revoke commands. Additionally, it enforces a MAC security policy on the subjects and objects under its control by associating them with sensitivity labels provided by the operating system, and utilizing operating system label-comparison functions to mediate access. Labelled objects include databases, tables, views, stored procedures, rows, messages, defaults, user datatypes, and rules. Subjects correspond to user DBMS sessions, and each has a sensitivity label range that determines the objects that it may observe and modify. The subject label range is always constrained by the user's operating system session label.

3.6 Multilevel network security systems

As an alternative to using MLS operating systems as the basis of security, we now examine the use of MLS as a foundation for network security. We consider two efforts that influenced this thesis, the seminal work by Rushby and Randell [71], and the work by Kang, Moore and Moskowitz [51]. The systems proposed by both efforts implement multilevel security and are primarily concerned with secure communication between computers in a network.

3.6.1 A distributed secure system

The focus of Rushby and Randell's work [71] was on constructing a distributed secure system, rather than a secure operating system. The proposed system, illustrated in figure 3.6, combines a number of different security mechanisms to provide a general-purpose distributed computing system.

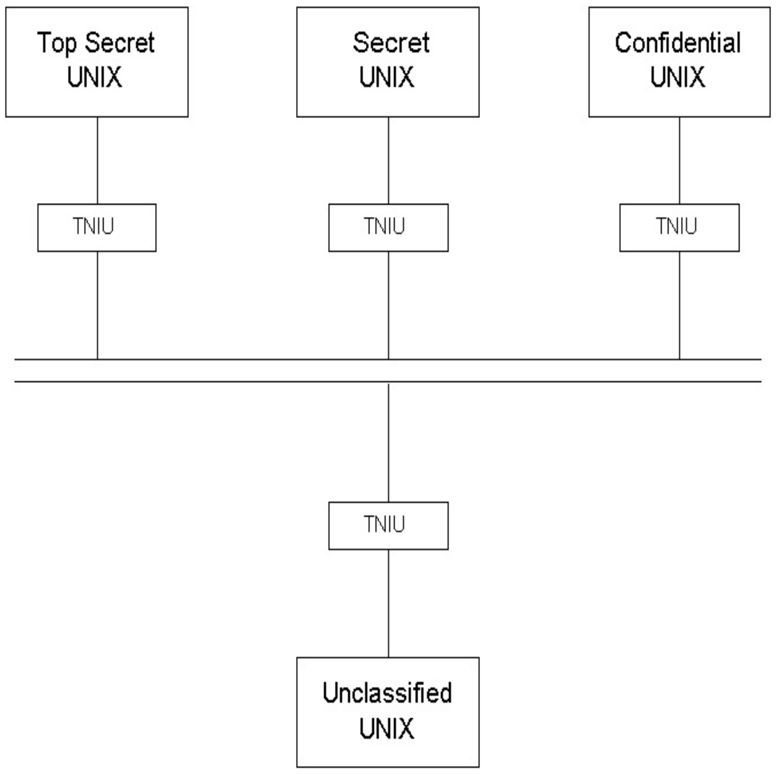

Figure 3.6: Conceptual structure of the distributed secure system.

The approach involves interconnecting small, specialised, trusted systems and a number of larger, untrusted host machines. Each untrusted host machine provides services to a single security level. The trusted components mediate access and communications between the untrusted hosts; they also provide specialized services such as a multilevel secure file store and a means for changing the security level to which a given host belongs. The approach requires no modifications to the untrusted host machines, allowing them to provide their full functionality and performance. Furthermore, the approach allows the mechanisms of security enforcement to be isolated, single purpose, and simple.

The approach to the design of the secure system is based on the key notions of *separation and mediation*, distinct logical concerns whose mechanisms must be kept distinct for ease of development and verification. The proposed system uses four different separation mechanisms: physical, temporal, logical, and cryptographic. *Physical* separation is achieved by allocating physically different resources to each security level and function. The authors suggest that trusted reference monitors can be used to control communication between the distributed components and to perform other security-critical operations. Due to the cost of providing physical separation for each security level and reference monitor, only the untrusted computing resources (hosts) and the security processors that house the trusted components are physically separate. *Temporal* separation allows the untrusted host machines to be used for activities at different security levels by separating those activities in time. The system state is re-initialised between activities belonging to different security levels.

The security processors in the system can each support a number of different separation and reference monitor functions, and also some untrusted support functions, by using a separation kernel to provide *logical* separation between those functions. The fourth technique, *cryptographic* separation, uses encryption and related (checksum) techniques to separate different uses of shared communications and storage media.

The four separation techniques provide the basis for a heterogeneous distributed secure system comprising both untrusted general-purpose systems and trusted specialised components, and to be useful it must operate as a coherent whole. To this end, the mechanism for providing security is built on a distributed system called Unix United. A Unix United system is composed of a (possibly large) set of interlinked standard Unix systems, or systems that can masquerade as Unix at the kernel interface level, each with its own storage and peripheral devices, accredited set of users, and system administrator.

A *secure Unix United* system is composed of standard Unix systems (and possibly some specialised servers that can masquerade as Unix) interconnected by a local area network (LAN). All the components of the Unix United system are assumed to be untrusted; the security of the overall system therefore does not depend on assumptions concerning their behaviour – except that the LAN provides the only means of communication. The consequence of not trusting the individual systems is that the unit of protection must be those systems themselves and not some sort of communal utility. Although there is no security *within* the individual Unix systems, the strategy of the approach is to enforce security on the communication of information *between* systems. To this end, a trusted mediation device called a trusted network interface unit (TNIU) is placed between each system and its network connection. The purpose of the TNIUs is to permit communication between machines belonging to the same security level. Controlling which hosts can communicate with one another is a reference monitor function, but because the LAN can be subverted or tapped, the TNIUs must also provide a separation function to isolate and protect the legitimate host-to-host communication channels. This separation function is provided cryptographically, with TNIUs encrypting all communications sent over the LAN.

Encryption is traditionally used to protect communications between parties with a shared interest in preserving the secrecy of their communications, but this is not the case here. Host machines are untrusted and may attempt to thwart the cryptographic protection provided by their TNIUs. For this reason, the encryption must be managed very carefully to prevent unauthorised communication between host machines, or between a host machine and a wiretapping accomplice.

3.6.2 NRL pump

Kang, Moore and Moskowitz [51] proposed the NRL pump as a device for enforcing multi-level security between LANs. It is a hardware device similar to the TNIU described earlier in section 3.6.1, and it is configured as a single device that interconnects a System-high LAN and a System-low LAN. In essence, the pump places a buffer between System-high and System-low, pumps data from System-low to System-high, and probabilistically modulates the timing of the acknowledgement from System-high to System-low on the basis of the average transmission times from the System-high LAN to the pump. The applications that have to interact in the System-low and System-high enclaves communicate with the pump through special interfacing software components, called wrappers, which implement the pump protocol. In particular, each wrapper is made of an application-dependent part, which supports the set of functionalities that satisfy application-specific requirements, and a pump dependent part, which is a library of routines that implement the pump protocol. Each message that is received and forwarded by the wrappers includes 7 bytes of header field, containing information about the data length, some extra header, and the type of message (data or control).

The pump can be considered as a network router. For security reasons, each process that uses the pump must register its address with the pump administrator, which is responsible for maintaining a configuration file that contains a connection table with registration information. The pump provides both recoverable and non-recoverable services [2]. Our discussions will concentrate on non-recoverable applications such as FTP.

The procedure for establishing a connection between System-low and System-high LANs through the pump is as follows. Initially, the System-low LAN sends a connection request message to the main thread (MT) of the pump, which identifies the sending System-

[2]Recoverability safety assumes that any message sent will be delivered to the System-high system, even if connection failures occur.

low process and the address of the receiving System-high process. If both addresses are valid (i.e., they have been previously registered in the configuration file managed by the pump administrator), MT sends back a connection valid message; otherwise it sends a connection reject message. In the first case, the connection is managed by a trusted System-low thread (TLT) and a trusted System-high thread (THT), which are created during the connection setup phase to interact with the System-low LAN and the System-high LAN, respectively. Registered System-high processes are always ready to accept a connection from the pump through the handshake mechanism described above. Once the new connection is established, the pump sends a connection grant message to both systems with initialisation parameters for the communication.

During the connection between System-low and System-high LANs, TLT receives data messages from the System-low LAN, then stores them in the connection buffer. Moreover, it sends back the acknowledgements (which are special data messages with zero data length) in the same order that it received the related data messages, by introducing an additional stochastic delay computed on the basis of the average rate at which THT consumes messages. On the other hand, THT delivers to the System-high LAN any data message contained in the connection buffer. The pump protocol also requires the System-high LAN to send back to THT the acknowledgement messages related to the received data messages. If the System-high LAN violates this protocol, THT aborts the connection. In such a case, as soon as TLT detects that THT has died, it immediately sends all the remaining acknowledgements and a connection exit message to the System-low LAN. Another special data message is connection close, which is sent at the end of a normal connection from the System-low LAN to the pump.

3.7 Database benchmarking

Benchmarks are vital tools in the performance evaluation and measurement of relational DBMSs. A *database benchmark* is defined as a standard set of executable instructions which are used to measure and compare the relative and quantitative performance of two or more database systems through the execution of controlled experiments [44]. Standard benchmarks such as the Wisconsin [12], TPC-A [79], TPC-B [80], TPC-C [81], and AS^3AP [82] benchmarks have been used to assess the performance of relational DBMS software. A wide variety of users have been dependent upon these benchmarks to select systems, to determine bottlenecks, and to verify technology improvement. The Wisconsin and AS^3AP benchmarks are widely considered as the standard relational query benchmarks [44], although the AS^3AP is also a complex mixed workload benchmark.

The Wisconsin Benchmark described in [12] was the result of the first effort to systematically measure and compare the performance of relational database systems with database machines. The benchmark is a single-user and single-factor experiment using a synthetic database and a controlled workload. It measures the query optimization performance of database systems with 32 query types to exercise the components of the proposed systems. The query suites include Selection, Join, Projection, Aggregate, and simple Update queries. The test database consists of four generic relations. Two data types, namely small integer number and character string are utilized. Data values are uniformly distributed. The primary metric is the query elapsed time. The main criticisms of the benchmark [40, 41] include the nature of single-user workload, the simplistic database structure, and the unrealistic query tests. A number of efforts have been made to extend the benchmark to incorporate the multi-user test. However, they have not received the same degree of acceptance as the original Wisconsin benchmark, except for an extension called the AS^3AP benchmark.

The ANSI SQL Standard Scalable and Portable (AS^3AP) Benchmark described in [82] models complex and mixed workloads, including single-user and multi-user tests, and operational and functional tests. There are 39 single-user queries consisting of Selection, Join, Projection, Aggregate, and Bulk Updates. The four multi-user modules include a concurrent random read test or pure information retrieval (IR) test to execute a one-row selection on the same relation, a concurrent random write test or pure on-line transaction processing (OLTP) test to execute a one-row update on the same relation, a mixed IR test, and a mixed OLTP test. The concurrent random read test measures the maximum number of concurrent users that can be handled by the system when a retrieval operation on the same relation is being performed. The concurrent random write test measures the maximum number of concurrent users that can be handled by the system when the same relation is being updated. The mixed IR test and the mixed OLTP test measure the effects of the cross-section of queries on the system with concurrent random reads or concurrent random writes. The test database consists of four generic relations, each having the same number of fields and the same number of records. The database scales up by increasing the number of records in each table. A number of data types are used, including long integer number, double precision floating point number, decimal number, money, datetime, fixed-length and variable-length character strings. Data values are created with uniform and non-uniform distributions. A new performance metric, the equivalent database size, is defined to measure the largest database size the proposed system can process within a 12-hour time limit. The benchmark tries to provide a balanced workload to test the system performance on utilities, access methods, query optimization, and concurrency control.

As one of the main objectives of our research is to instrument a performance study of a prototype MLS/DBMS, we considered 1) using one of the standard DBMS benchmark suites such as the Wisconsin and AS^3AP Benchmarks, since there is no publicly available MLS/DBMS benchmarking suite, or 2) implementing a prototype and instrumenting performance experiments with real rather than synthetic variables. We chose the latter approach because, although the standard benchmarks are generally considered sufficient for benchmarking standard DBMSs, they were not designed to benchmark MLS/DBMSs. Standard DBMS workloads are not representative of MLS/DBMS workloads which must, in addition to such factors as database size, and number of concurrent users, consider data and users at different security levels. A major factor that impacts the performance of MLS/DBMS systems is the overhead imposed by the processing algorithms that handle these multilevel data and users. Existing benchmarking suites do not simulate the impact of this processing overhead. In the following subsection, we consider an MLS/DBMS performance study that was instrumented to measure the cost of alternative query processing strategies; this study was not undertaken using a DBMS benchmark suite.

3.7.1 A Trusted Database Management System (TDBMS) performance study

A performance study by Thuraisingham and Kamon [77] was based on a *trusted database management system* (TDBMS). The authors implemented and benchmarked a TDBMS based on the distributed architecture approach recommended by the Woods Hole study group [6]. The objective of this study was to validate the security policy for the query operation and to analyze the performance of query processing algorithms in the TDBMS implementation. This was the first major performance study of an MLS/DBMS. A commercial SYBASE DBMS running on a Berkeley Unix 4.2 operating system was used as the back-end DBMS for the study. The database contained two relations which may be horizontally partitioned across security levels. The Select-All and Join query operations were used to interrogate the database as the number of tuples in the relations were varied. The authors considered a variety of query processing strategies for their cost and suitability. The performance measurements were based on the response time of each query; this metric is also used in this thesis. Some of the conclusions from the study were: 1) it was more beneficial to perform Join operations at a back-end machine rather than the front-end; 2) performance was unaffected by polyinstantiation as long as lower level polyinstantiated tuples were not removed from a user's view.

3.8 Summary

A U.S. Air Force sponsored summit to address issues relating to MLS/DBMS design proposed that different architectures be considered for building MLS/DBMSs. These include the integrity lock architecture, kernelized architecture, replicated architecture, and trusted subject architecture. Each of these architectures provide a way for storing data and controlling access to it. A number of MLS/DBMS prototypes were developed over the past two decades based on proposals made at the summit, including the SeaView prototype, the LDV prototype, the ASD prototype, and the SWORD prototype. None of these prototypes became a commercial product, but some commercial database products promising varying degrees of security were also developed, shipped, and withdrawn during the same time period. These include Trusted Oracle 7, Ingres, Informix, and Sybase.

The prototypes and systems differed in the granularity of objects that they supported. For example, SeaView supported labelling at an individual attribute value level, LDV supported tuple-level labelling, SWORD supported element-level labelling, and in ASD the security object was a materialized view. Some of the commercial systems also supported security labelling at the relation level or even the database level.

Benchmarks are crucial tools in the performance evaluation of database systems. Relational database benchmarks, such as the Wisconsin and the AS^3AP, have been widely used to develop performance models, to compare alternative designs, to pinpoint system bottlenecks, to select software, and to predict system behaviour. They serve as indispensable tools in assisting academics, practitioners, programmers, benchmarkers, and even managers, to validate database research results, to verify software prototype improvement, and to facilitate the systems selection and procurement process.

A notable performance benchmark study of an MLS/DBMS system was conducted by Thuraisingham and Kamon. This study, for the first time, gave some measure of the performance overhead imposed by MLS processing on standard database operations.

Chapter 4 presents a hybrid conceptual model for mandatory access control including a description of its salient features.

Chapter 4

A Conceptual Model for Access Control

4.1 Introduction

This chapter presents a conceptual description of a MAC model for the protection of data in MLS/DBMSs. Access control is governed by axioms slightly stricter than those of the BLP model. A TCB enforces the MAC policy, mediating the access requests of subjects to objects such that there is no unauthorised disclosure. Although the emphasis of the model is on the protection of the confidentiality of information, a number of integrity issues that arose as a result of the protection mechanisms of the model are also addressed. Most of the existing MLS/DBMS models were designed to be incorporated into a DBMS at the design phase of development rather than post-development. The model described in this chapter is proposed specifically to address the needs of an organization that wishes to adapt its existing COTS DBMS to provide MLS at a reasonable cost. Earlier in chapter 3 we described two approaches for representing and controlling access to data, the polyinstantiation approach and the SWORD approach. Our model incorporates features from both approaches but draws more heavily from the SWORD approach, specifically from its insert low approach and its concept of place holders.

The following section describes the basic properties of the proposed model and where it fits between the polyinstantiation approach and the SWORD approach. It also describes how the proposed model incorporates features from the two camps. Section 4.3 describes the BLP axioms for controlling access to objects and provides a description of the interpretation of these axioms in data processing operations including insert, update, delete, and select. Section 4.4 summarizes the chapter.

4.2 Basic properties of the model

The proposed model is largely oriented towards models that represent data 'truthfully', e.g., the SWORD approach, rather than those that are oriented towards representing data 'honestly', e.g., the Jajodia-Sandhu model. Although the model uses the Jajodia-Sandhu definition of a multilevel relational model, its operational semantics is based largely on the insert low approach advocated by SWORD. The choice between these two approaches was determined in part by integrity and performance considerations. We wanted a model that incorporated the desirable features of the SWORD and the polyinstantiation approaches, without inheriting their problems. For example, a problem with adopting the polyinstantiation approach in its entirety is its lack of support for element-level constraint enforcement, and its inability to guarantee that the values in a column are unique. Furthermore, using the polyinstantiation approach would inevitably introduce performance penalties because any increase in the number of security levels will result in a corresponding increase in the number of instances that must be processed in response to a query.

Although the model inherits most of the query processing techniques of the SWORD approach, including the concept of place holders, its data labelling feature is not only at the element-level, and is not augmented with a table classification as is the case in SWORD. Furthermore, in adopting the insert low approach, the model does inherit the potential problem of a denial of service attack that could be launched by a low subject that overclassifies a data object.

Earlier in sections 3.4.5 and 3.6.2, we described Purple Penelope and the NRL pump, respectively. Unlike Purple Penelope, our approach does not support discretionary and role-based access control policies — a main feature of Purple Penelope. In addition, our approach, in contrast to Purple Penelope, does not permit users at all security levels to label data objects. In the case of the NRL pump, as an application-independent device that enforces access control 'remotely', it contrasts with our approach which enforces access control locally — at the DBMS layer. Furthermore, because the NRL pump was not designed to enforce security within the individual System-high LAN and System-low LAN, there is no mechanism to prevent access control violations within the individual LANs if the pump is bypassed.

4.2.1 Multilevel relations

The model represents a multilevel relation as sets of relation fragments, with each set comprised of fragments with the same security classification. Relation fragments in a set share the same schema, and are generated by the application of decomposition algorithms to a larger relation in accordance with the fragmentation rules described earlier in section 2.6. Therefore, all the relation fragments are disjoint and are not replicated at each security level. The model, like the Jajodia-Sandhu model, supports classification at the tuple, attribute, and element levels of classification, but differs from the latter's support for polyinstantiation.

Read and write operations on the relations are controlled and restricted to those satisfying the 'No Read-Up' and 'No Write-Down' axioms of the BLP model. A null value in a field indicates that the subject is not cleared to observe that field. The same semantic obtains in the Jajodia-Sandhu model and the SWORD approach.

Earlier in section 2.4.2.3 we defined a *trusted subject* as a subject that may violate the BLP security policy. The same definition holds for the proposed model. However, the proposed model uses the current security level f_c to escape from the strict restrictions imposed by the *-property of BLP. The current security level allows a subject to be temporarily downgraded before it can execute an insert operation.

4.3 Access to multilevel relations

This section describes the axioms for mediating the access of subjects to objects, and how these rules translate into specific database operations. The two axioms that regulate the access of the subjects in the database are:

1. **Write access**
   A database insert and delete request is only granted to subjects at System-low. For update requests to be permitted, the subject must have a clearance corresponding to the object's classification.
   The write access rule corresponds to the No Write-Down principle in BLP. However, it is stronger than the BLP constraints in that insert requests are only permitted from subjects at System-low.

2. **Read access**
   A database read request from a subject at level l is only permitted if the object requested is at level l and below.
   The read rule corresponds to the BLP No Read-Up principle.

In the following sections, we describe the translation of the write and read axioms above into actual operations on objects. Write operations supported include insert, update, and delete. The read operation supported is select.

4.3.1 Insert operation

Rows are inserted using place holders, as described earlier in section 3.4.4. A System-low subject is permitted to label a new data object with any classification level that dominates his/her clearance. The Jajodia-Sandhu model, like the BLP model, permits insertions by users only if the classification of the object dominates the clearance of the subject. The proposed model, like the SWORD approach, considers this simple test to be insufficient for the purposes of data integrity. It discourages blind-writes by offering a stricter interpretation of the BLP model NWD property. It permits insertions only by subjects at System-low; however, this alone cannot guarantee the integrity of data. Users cleared at System-high must downgrade their clearance to System-Low in order to perform this operation. The insert operation has the following form:

INSERT INTO *table_name* **VALUES** $(value1, value2, \ldots)$

Depending upon which fragmentation scheme is being implemented, a specific $value_i$ could represent a security classification value.

4.3.2 Update operation

A subject may update an object only if the clearance of the subject is equal to the classification of the object. This is a stricter interpretation of the BLP NWD property. It differs from the Jajodia-Sandhu model and the SWORD approach because it only permits subjects to update objects at their own security level and not at a higher level. The Jajodia-Sandhu model and SWORD both permit update operations only if the classification of the data object dominates the clearance of the subject. The update operation has the following form:

UPDATE *table_name*
SET *attribute_name* = new_value
WHERE *selection_condition*

4.3.3 Delete operation

A subject may delete an object only if the clearance of the subject is at System-low. Once again, this operation, for the purposes of data integrity, is stricter than is required under the BLP and Jajodia-Sandhu models. In BLP, a subject only needs to be dominated by an object in order for a delete operation to be permitted. The delete operation is based on that of the SWORD approach, but without the alternative requirement in SWORD for a table classification to be dominated. The update operation has the following form:

DELETE FROM *table_name*
WHERE *selection_condition*

4.3.4 Select operation

A subject may retrieve an object only if the clearance of the subject dominates the classification of the object being requested. This rule is in agreement with the BLP model NRU property. It is also consistent with the interpretations provided by the Jajodia-Sandhu model and SWORD approach. The update operation has the following form:

SELECT *attribute_list*
FROM *table_list*
WHERE *selection_condition*

The attribute list specified never contains the security classifications of the corresponding attribute values.

4.4 Summary

This chapter gives an informal description of a conceptual model for enforcing mandatory access control in an MLS/DBMS. The model is largely oriented towards representing data 'truthfully', as in the SWORD approach rather than representing data 'honestly', as in the Jajodia-Sandhu model. The model uses the Jajodia-Sandhu definition of a multilevel relational model, but its operational semantics is largely based on the insert low approach advocated by SWORD.

The model complies with the BLP NWD and NRU security properties but, in the case of the former, it offers a stricter interpretation of the rules. The primary concern of the model is the confidentiality of information, but it also addresses, in a limited way, some integrity issues pertaining to the insertion and deletion of data objects.

Implementing the model described in this chapter requires the identification of a suitable target DBMS that can be augmented with the security policy captured by this model. Chapter 5 presents the architecture of such a system — the Stargres distributed database management system, a COTS DBMS product, including a description of its constituent facilities and modules.

Chapter 5

Software Architecture of a COTS DBMS

5.1 Introduction

This chapter describes the architecture of the Stargres distributed database management system. This DBMS is one flavour of the open source Ingres distribution. Stargres is comprised of a Distributed server component and the DBMS subsystems. The Distributed server is a data manager that adds the functionality of a distributed relational DBMS to Stargres. This chapter presents Stargres mainly from the viewpoint of the facilities and modules that comprise each of its subsystems, our intention being to disentangle the code, particularly the Distributed server code, and the interdependencies between its modules before embarking on any kind of adaptation effort.

In the next section we briefly explain why we chose Stargres as a candidate for adaptation. Section 5.3 describes the software architecture of the Distributed server through its component modules. Section 5.4 presents the software facilities of the underlying DBMS and brief descriptions of their component modules. In section 5.5 we show how tables are created in Stargres. Section 5.6 summarizes the chapter.

5.2 The Stargres distributed DBMS

Before we chose the Stargres DBMS as our candidate for adaptation, we also considered other distributed DBMSs, namely Mnesia[1], and Mariposa[2]. Mnesia was considered unsuitable because it did not support the types of fragmentation schemes advocated in our research; furthermore, it was developed primarily to address the needs of the telecommunications industry, and the source code was not completely open. Mariposa was considered

[1]Mnesia is a telecommunications industry deductive DBMS developed by Ericsson^{TM}.

[2]Mariposa is a wide area network distributed DBMS developed at the University of California at Berkeley.

unsuitable because each of its objects is constantly migrating and does not have a ‘fixed home’; moreover, the latest release of Mariposa was found to be unstable in our operating system environment.

Stargres is a large software system, which has of the order of hundreds of thousands of lines of code. Its support for fragmentation and query decomposition through its Distributed server feature set made it an ideal choice for implementing our design. Furthermore, its collection of APIs makes handling of host application calls highly efficient. Although we did not engage in a full-scale reverse engineering effort in order to disentangle Stargres, we made extensive use of the documentation made available by its developers and a reverse engineering toolkit called SWAG[3] to determine the modular organisation and their interdependencies.

The Stargres DBMS uses the star query language (StarQL), an extension of the ANSI standard SQL [56]. StarQL was specially developed for Stargres and supports the full range of features provided by standard SQL. StarQL also provides some additional features and commands that extend the functionality of standard SQL; these will be introduced in the following sections and in Chapter 6 as our description of the Stargres architecture proceeds.

5.3 Software modules of distributed server

Figures 5.1 and 5.2 illustrate the high-level and detailed software architecture of the Stargres DBMS respectively. In this architecture, the Distributed server component receives all requests for data from users and forwards the requests to authorised local DBMSs; it never directly accesses the data fragments stored in databases. A local DBMS accesses its database and returns the results to the Distributed server which, in turn, forwards it to the user. Modules belonging to the Distributed server perform all the central coordination and global processing tasks associated with managing the distributed data. The DBMS modules perform all the local processing tasks. This section will describe the facilities and modules provided by the Distributed server component; these include: the Global DDL compiler (GDC), the Distributed query handler (DQM), the Query transformer module (QTM), the Distributed query optimizer (DOPM), the Distributed query execution engine (DQEM), the Remote access module (RAM), and the Coordinator database module (CDM).

5.3.1 Global DDL compiler (GDC)

Administrative queries are routed to the GDC module by the Query router module (QRM) for compilation. Global DDL statements are compiled and forwarded directly to the DQEM

[3]SWAG is a Toolkit developed by the Software Architecture Group at the University of Waterloo in Canada for extracting and presenting software architectures; it supports the extraction of C, C++ code.

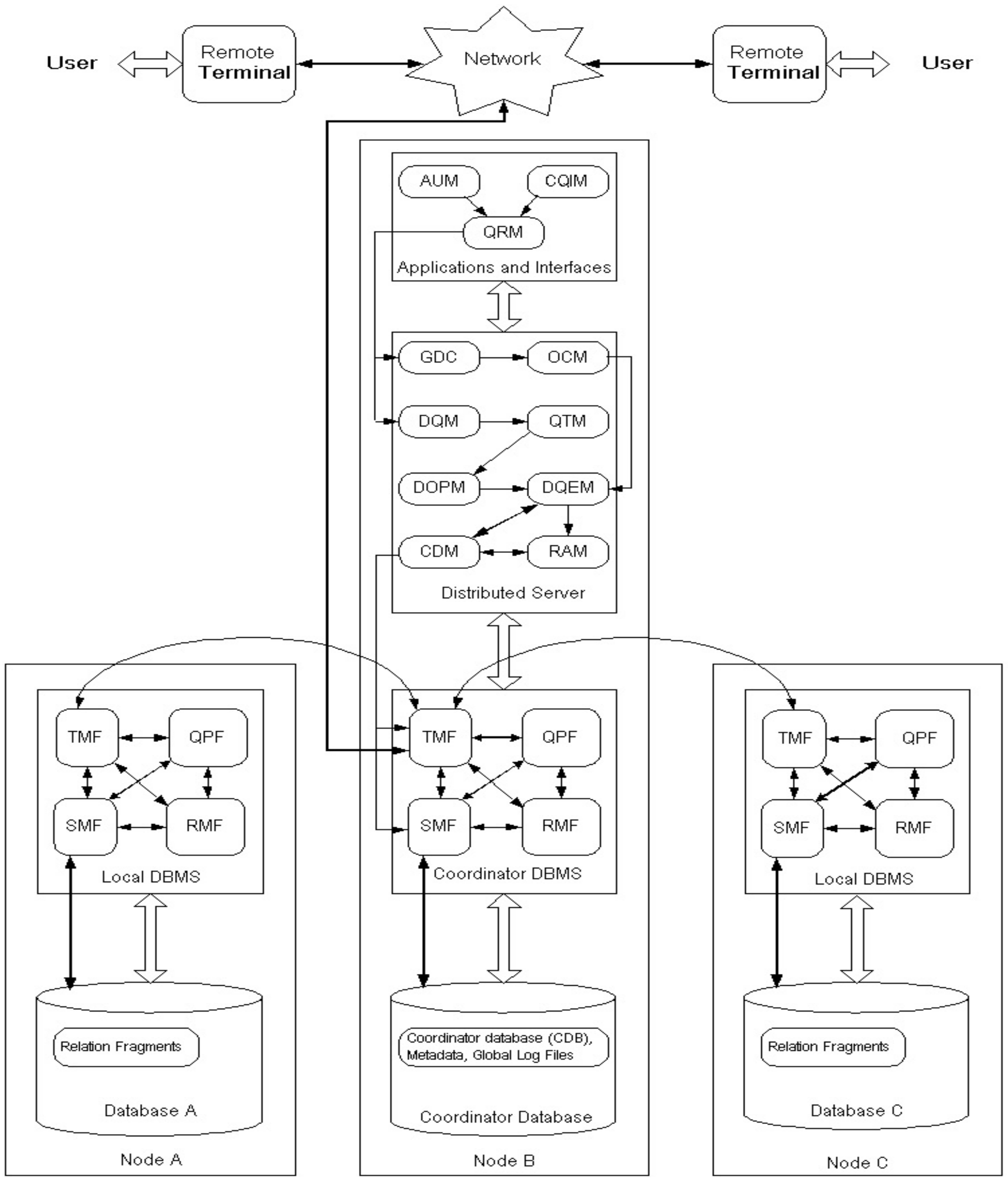

Figure 5.1: Software architecture showing two nodes connected by a third node running the Distributed server component. The Distributed server uses a dedicated coordinator DBMS and database to manage the nodes in its domain. The system provides interfaces that support programmatic and interactive queries. Application programs and interactive users connect to the system from remote workstations or via a virtual private network (VPN).

module for execution. Each of these statements affect the structure of relation fragments maintained in the nodes or the profile of objects such as user accounts and nodes. The CREATE DATABASE, CREATE TABLE, and CREATE USER statements add new definitions of either databases, tables, or users to the coordinator database (CDB) catalog. The ALTER TABLE command changes the structure of an existing table by updating the global schema in the CDB catalog; it is used to add or modify columns, change the type of existing columns, or rename columns or the table itself. The REGISTER node statement registers a new node with the Distributed server and updates the CDB catalog; the converse of this command is the REMOVE node statement.

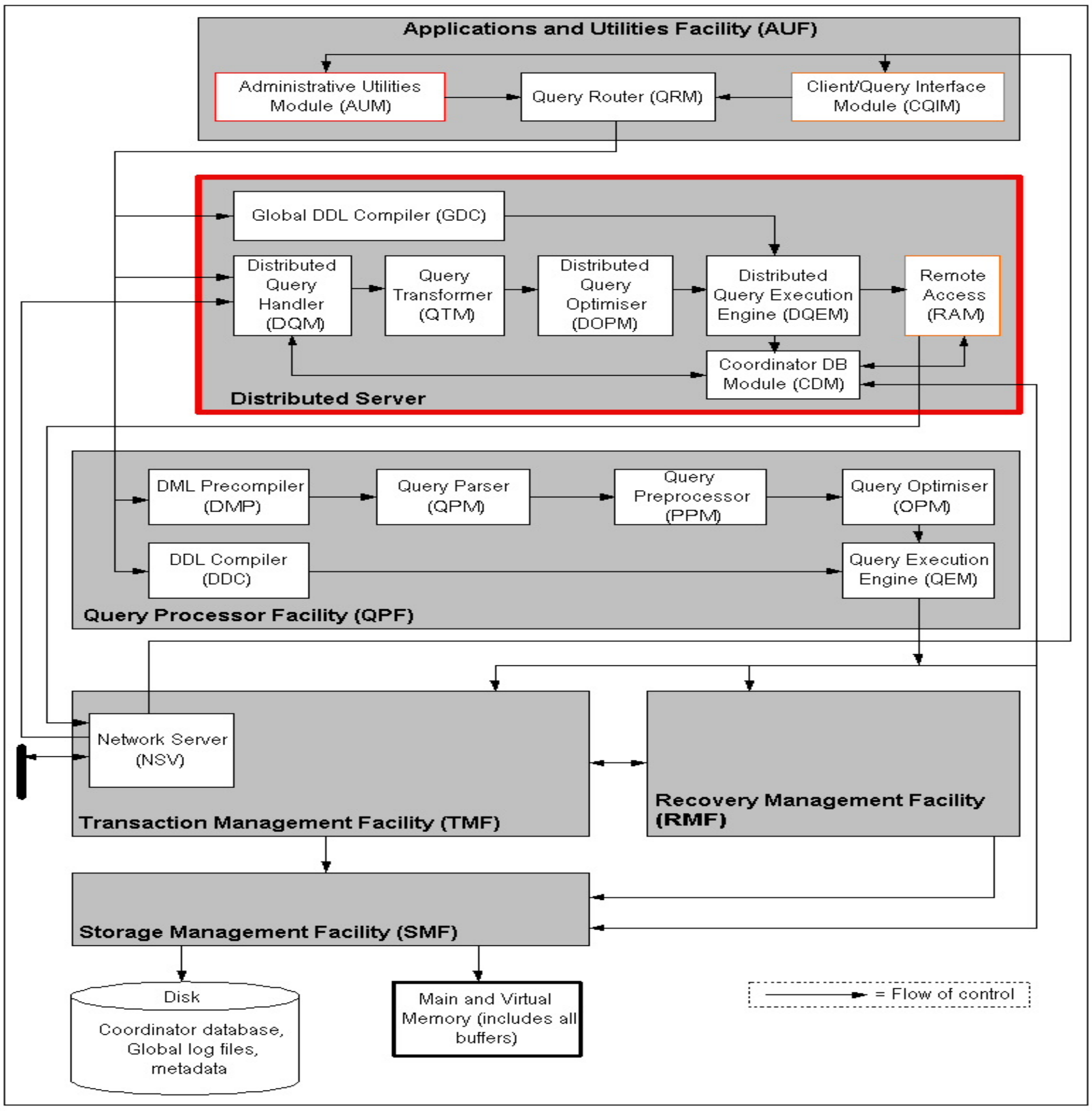
Applications and Utilities Facility (AUF)
Administrative Utilities Module (AUM)
Query Router (QRM)
Client/Query Interface Module (CQIM)
Global DDL Compiler (GDC)
Distributed Query Handler (DQM)
Query Transformer (QTM)
Distributed Query Optimiser (DOPM)
Distributed Query Execution Engine (DQEM)
Remote Access (RAM)
Coordinator DB Module (CDM)
Distributed Server
DML Precompiler (DMP)
Query Parser (QPM)
Query Preprocessor (PPM)
Query Optimiser (OPM)
DDL Compiler (DDC)
Query Execution Engine (QEM)
Query Processor Facility (QPF)
Network Server (NSV)
Transaction Management Facility (TMF)
Recovery Management Facility (RMF)
Storage Management Facility (SMF)
Disk
Coordinator database, Global log files, metadata
Main and Virtual Memory (includes all buffers)
= Flow of control

Figure 5.2: Detailed software architecture of the Distributed server node depicted in figure 5.1. It shows the interaction between the modules of the Distributed server and the host DBMS. The Distributed server adds the GDC, DQM, QTM, DOPM, DQEM, RAM, and CDM modules to the DBMS layer below.

5.3.2 Distributed query handler (DQM)

This module intercepts incoming global queries from the network server module (NSV) and executes a simple syntax and verification check; it also assembles partial responses received from local nodes. The module verifies the existence of all fragments referenced in the query before forwarding it to the QTM module for the next phase in processing.

5.3.3 Query transformer module (QTM)

The QTM decomposes global queries against relations into sub-queries against relation fragments. It constructs query responses using the Union, Join or a combination of Union and Join to combine the relation fragments into a single relation. The module generates sub-queries against relation fragments in two steps. First, it maps the distributed query into sub-queries and substitutes each distributed relation by its reconstruction query expression; second, it simplifies and restructures the fragment query according to the same rules used during the query decomposition phase.

5.3.4 Distributed query optimiser module (DOPM)

The DOPM gets the sub-queries from the QTM module as input and generates a global query execution plan as an output. The query execution plan identifies the sites where the query needs to be executed, the partial order in which the query gets executed, and where the final result needs to be sent. The data transfer operations for executing the global query are specified in the execution plan.

5.3.5 Distributed query execution engine (DQEM)

The DQEM module provides functions that are used for the creation of relation fragments, the updating or deletion of several databases within a transaction, and the transmission of sub-queries to local sites for processing. A record of the transactions being processed by this module is written to the coordinator database through the CDM which keeps track of the completion status of each transaction. All newly created relations are registered with the coordinator database through the CDM. The DQEM references the global CDB catalog maintained in the coordinator database in order to successfully execute transactions. Local transaction manager modules must check this catalog before beginning an operation.

5.3.6 Remote access module (RAM)

The RAM uses information stored about nodes by the CDM module during the registration of nodes, and the creation of databases, tables, and user accounts. This information includes the node name, database name, table name, and the table location, as necessary in order to direct queries and to distribute fragments to relevant nodes. It utilises a mixture of graph and stack data structures to help distribute relation fragments, and tree data structures for its distributed query processing. It contacts the NSV module in the Transaction Management Facility to open connections to the nodes that the user is authorised to contact during a particular transaction.

5.3.7 Coordinator database module (CDM)

The CDM manages the CDB which maintains the catalogs that the Distributed server uses to keep track of all the databases in the cluster. On receipt of a query, the DQM module sends a request to the CDM to verify the existence of the tables referenced. If the table verification is successful, the Distributed server then forwards the query to the nodes for processing. Although the coordinator database is similar to any other database, it is intended for use solely by the Distributed server. This module also manages the CDB catalog containing the names of all relation fragments, their locations, and their status (online or offline). The CDM also maintains the global log file on transactions; these files are used to facilitate recovery operations. The coordinator database, managed by the CDM, stores the mapping from global relation to fragments, and the mapping of fragments to sites. The DQM, DQEM, and RAM modules access the database and its catalog through the CDM. The CDB catalog is not replicated.

5.4 Software facilities of the underlying DBMS

Figure 5.1 also illustrates the software architecture of the underlying DBMS. The Distributed server relies on the functionality of this layer. Figure 5.2 is an expanded view of the high-level architecture described in figure 5.1; it shows the software modules in all the facilities except the Storage and Recovery Management Facilities, and portrays a view of the control flow within the DBMS system. A pipeline architecture, described by Garlan and Shaw [36], is used in the Query Processor Facility between the Data Manipulation Language Precompiler (DMP) and the Query execution engine. For the sake of simplicity, information flows back up the architecture have been omitted, and should be deemed as implicit. For example, calling a simple SQL command, such as SELECT $\star$, would require information to be brought back up the system. This section describes the Applications and utilities facility (AUF), the Query processor facility, and the Transaction management facility.

5.4.1 Applications and utilities facility (AUF)

The AUF accepts queries either interactively from a user or from an application program, and screens the query for 'prohibited administrative operations' and syntax errors. In the interactive mode, a user either enters a single SQL statement at a console using SQL syntax such as SELECT, FROM, WHERE, AND, or enters several statements enclosed by the keywords "begin" and "end" to get multiple results. In the application program mode, an embedded SQL application submits a query through its API. SQL statements embedded in application programs are prefixed by the EXEC SQL command, so that they are easily distinguishable from the host language statements, and are terminated by a special symbol such as a semicolon. Three modules in AUF handle the different kinds of queries that are submitted to the DBMS. These modules can be seen in the layered DBMS architecture diagram in figure 5.2, and include the Administrative utilities module (AUM), the Query router module (QRM), and the Client/Query interface module (CQIM) (see below).

Administrative utilities module (AUM)

The AUM, in addition to performing syntax checks, provides a number of utilities for carrying out administrative tasks such as registering nodes, creating user accounts, creating or dropping databases and tables, schema definition, shutting down the server, and so forth. Except for the CREATE table command, which is executable by all users, all the other DDL commands require administrative privilege. In Stargres, two user accounts are automatically created as part of the installation process and assigned to the DBA group. The DBA group is one of the most important default groups that exists in Stargres; it gives all administrative privileges to users who are members of this group, thus enabling them to perform the administrative tasks.

Query router module (QRM)

The QRM directs queries to the appropriate pipeline for processing. When a request is received, the QRM examines the query for a local or distributed switch statement. If a local switch statement is encountered, the query is examined again to determine its type, and then routed. DML queries are routed to the DMP module, and DDL queries are routed to the DDC module. If a distributed switch statement is encountered, the query is routed to the GDC module if it is a DDL query, otherwise it is routed to the DQM module.

Client/Query interface module (CQIM)

The CQIM provides the interface that facilitates interaction between users and the DBMS. The module also imposes restrictions on the types of commands that can be executed by non-administrative users; it enforces this constraint by responding with an error message whenever it receives a request to execute the REGISTER node, REMOVE node, CREATE database, DESTROY database, CREATE user, or DROP user command from a user. The concept of 'prohibited administrative operations' takes account of environments where the need exists to reserve certain types of operations for the administrator only.

5.4.2 Query processor facility (QPF)

The vast majority of interactions with the system occur when an application program or user wishes to query stored data. These queries, which are specified using a DML, are parsed and optimized by the query processor facility. This facility is represented as a pipeline and filter architecture, where the result of the previous component becomes an input or requirement to the next component. A query progresses through the DMP module, the DDL compiler (DDC), the query parser (QPM), the query preprocessor (PPM), the query optimiser (OPM), and the Query execution engine during processing.

5.4.3 Transaction management facility (TMF)

The purpose of this facility is to ensure that a transaction is logged and executed atomically. It does so with the aid of the concurrency control manager (CCM), the transaction manager (TMM), and the log manager (LMM). The transaction manager is also responsible for resolving any deadlock situations that occur and for issuing the COMMIT and the ROLLBACK SQL commands. Furthermore, this facility is also responsible for servicing connection requests to remote databases; it uses the NSV module to add the network communication element to the architecture. The NSV is running on each node and is responsible for opening and closing connections.

5.5 Database creation in Stargres

The CREATE database command is used by the RAM for creating Stargres databases; it has the form: CREATEdb <database name> (<fragmentation option>). The database name specifies the name of the database to be created. The name must be unique among all Stargres databases in a particular installation. The fragmentation option, tuple, attribute, or element, must be specified as a parameter in the CREATEdb statement. An error message is returned if the database creation is unsuccessful. We give a more detailed treatment of the CREATEdb statement in Chapter 6.

5.6 Table creation in Stargres

A CREATE table command is used by the RAM for creating and distributing relation fragments in each of the fragmentation schemes; it has the form: CREATE table <table name> (<attribute$_i$> <data type> <distribution criteria>, . . . , <attribute$_n$> <data type> <distribution criteria>). The *distribution criteria* parameter is a rule that is specified on one or more attributes of a relation schema when the schema is defined. This rule is expressed such that relation fragments can be distributed to designated nodes based on the value of specified attributes. The *distribution criteria routine* of the Distributed server uses this rule in deciding where to ship fragments; the creation of a table will fail if no rule is defined.

Stargres does not define distribution criteria on any attribute that is designated as the primary key, as it associates each attribute with the designated primary key attribute. A '$\star$' parameter in the specification of the distribution criteria is used to indicate that all values including 'nulls' of an attribute will be treated as if they were the same value for the purposes of distribution. The distribution criteria are stored in the CDB catalog along with relation schemas. In contrast to relation schemas which are stored globally, relation instances are stored at the various sites maintaining fragments.

The manner in which tables are created in Stargres differs for each fragmentation scheme. The following examples describe the command for creating tables; they are illustrated using the Department relation shown in Appendix A.

Tuple-level table creation

Tuple fragments are distributed to different nodes based on the distribution criteria specified on an attribute. Distribution criteria can only be specified on one attribute during the creation of a table. To send any tuple fragment with a Manager value of Cantor to Site-A and a Manager value of Gerald to Site-B, the required command has the form:

- **Command:** CREATE table Department (DN NUMBER(4) WITH
  (Dept_Name CHAR(25),
  Manpower CHAR(10),
  Dept_Location CHAR(25),
  Manager CHAR(25)),
  @ Site-A IF Manager = 'Cantor',
  @ Site-B IF Manager = 'Gerald');

Attribute-level table creation

Attribute fragments are distributed to relevant nodes based on the distribution criteria specified on the attributes of a relation during the creation of a table. The distribution criteria routine examines the values of each attribute to determine if they satisfy the distribution criteria. The specification of the distribution criteria on attribute values rather than attributes can conceivably result in fragments that are not vertical fragments, because of errors that arise during the specification of distribution criteria. A non-vertical fragment will result if the distribution criteria refer to a specific attribute value (e.g., Manpower CHAR(10) @ Site-B IF Manpower = '15') instead of using the '$\star$' placeholder. Although stored procedures can be used to reduce the possibility of these errors, they do not provide a robust solution. To send the attribute fragments Dept_Name to Site-A, Manpower to Site-B, Dept_Location to Site-C, and Manager to Site-D, the required command has the form:

- **Command:** CREATE table Department (DN NUMBER(4) WITH
  (Dept_Name CHAR(25) @ Site-A IF Dept_Name = '⋆',
  Manpower CHAR(10) @ Site-B IF Manpower = '⋆',
  Dept_Location CHAR(25) @ Site-C IF Dept_Location = '⋆',
  Manager CHAR(25) @ Site-D IF Manager = '⋆'));

Element-level table creation

The distribution of element fragments to nodes depends on the distribution criteria specified on the attributes of a relation; this rule must be specified on one or more attributes during the creation of a table. Here, the distribution criterion is similar to that used in the attribute-level example. To send an element fragment with a Dept_Name value of 'Radiology' to Site-A, a Manpower value of '15' to Site-B, a Dept_Location value of '512 West Wing' to Site-C, and a Manager value of 'Cantor' to Site-D, the necessary command has the form:

- **Command:** CREATE table Department (DN NUMBER(4) WITH
  (Dept_Name CHAR(25) @ Site-A IF Dept_Name = 'Radiology',
  Manpower CHAR(10) @ Site-B IF Manpower = '15',
  Dept_Location CHAR(25) @ Site-C IF Dept_Location = '121 South Wing',
  Manager CHAR(25) @ Site-D IF Manager = 'Cantor'));

5.7 Summary

In this chapter we have presented a description of the Stargres DBMS. The Stargres DBMS is made up of two major subsystems: the DBMS, and the Distributed server; each of these components is further sub-divided into facilities which consist of individual modules that perform specific processing functions. The Distributed server component comprises a number of modules including the GDC, DQM, QTM, DOPM, DQEM, RAM, and CDM. The underlying DBMS server also presents the Distributed server layer above with critical facilities such as the AUF, QPF, and TMF.

The next chapter describes how the Stargres DBMS was adapted to provide multilevel security. It focuses on the critical modules of the DBMS and the Distributed server.

Chapter 6

Adaptation of the Stargres DBMS

6.1 Introduction

This chapter presents the technical details of the Stargres COTS DBMS components that were adapted and systematically integrated with a computer cluster to enforce multilevel security. We refer to this integration of software and hardware components as the Multilevel Stargres (MST) prototype. Our description will focus mainly on the AUF and Distributed server modules that we adapted. The AUF and Distributed server modules were augmented with MLS code such that the rules for creating nodes, users, databases, relations, and for distributing relation fragments relied on a security level parameter. We use the term 'node' to refer to a single-level machine in the cluster, and the term 'Distributed server node' to refer to the node running the Distributed server software (see figure 5.1) in Chapter 5.

Section 6.2 presents an architectural overview of the prototype. Section 6.3 describes how the Distributed server was augmented with MLS information. In section 6.3.1, we describe how the Distributed server was adapted to capture and store MLS information. Section 6.3.2 describes how the prototype uses MLS information in registering nodes. Section 6.3.3 describes how MLS information is used to create user accounts. Section 6.3.4 describes how databases are created in MST. Section 6.3.5 describes how tables are created using MLS information. Section 6.3.6 describes the command used for altering tables. Section 6.3.7 describes the processing of Select, Join, Insert, Update, and Delete queries in all three fragmentation schemes. Section 6.4 summarizes the chapter.

6.2 Architectural overview

The MST prototype is based on the kernelized architecture approach described in section 3.3.2. The prototype consists of a LAN cluster of single-level machines, each containing relation fragments of the same security classification. A designated machine, running Distributed server software, provides the interface for programmatic and interactive user queries. Programmatic applications and interactive users access the Distributed server from remote workstations via the LAN or virtual private network (VPN). The Distributed server is the only trusted component, it enforces MLS control by directing queries only to authorised nodes (i.e. the set of nodes that a user is authorised to access), and by denying all connection requests to unauthorised nodes (i.e. the set of nodes that a user is not authorised to access). It does this by comparing the user's clearance with the levels of the target nodes (i.e. the set of nodes containing the fragments referenced in a query), and dropping all connection requests to nodes with classification levels higher than the user's clearance if the request is for a 'read operation' such as a SELECT SQL statement. All connection requests to nodes with classification levels lower than the user's clearance are also denied if the request is for a 'write' operation such as an INSERT SQL statement.

At the highest level of abstraction, the two software subsystems of the MST prototype include the DBMS and the Distributed server. These subsystems also consist of facilities, which in turn are made up of modules. We will focus our discussion on those facilities and modules that form the nucleus of the prototype — especially the Distributed server modules; this by no means diminishes the importance of the other facilities and modules which also contribute in varying measures to the operation of the prototype.

6.3 Augmenting the distributed server with MLS information

Our adaptation focused on the AUM and RAM. By default, these modules do not capture or utilise security levels for nodes or users during their respective processing operations. However, in order to deliver a robust MLS system, these critical modules had to be adapted to capture, store, and utilise security level information in addition to any other processing-specific information that may be required. The AUM was modified so that a security lattice can be defined by the administrator using a DEFINE lattice command. The AUM also requires the administrator to provide a security level when the REGISTER node or CREATE user command is issued. The security level field cannot be empty; it must contain an element from the lattice of security labels in use. Whenever a lattice element is entered as a parameter in the REGISTER node or CREATE user command, a validation routine compares the element with a set of valid security labels stored in the CDB, and an error message is returned if the label entered is invalid.

The administrator is responsible for assigning security levels to all nodes and users. A security lattice must have been defined in the system before the creation of a database or the registration of a node can proceed. The lattice implements a partial ordering $\leq$ on any set of elements of a newly defined data type called LABEL. Only elements from one security lattice can be in use at any one time in a database. In Figure 2.2(B) we presented an example of a lattice for a firewall; the examples in this chapter will be illustrated using this lattice.

The DEFINE lattice routine specified in Appendix B, prompts the administrator to provide a set of classification labels representing the elements of the security lattice. These elements, of type LABEL, are stored in array structures. The ordering of the elements in the arrays is not related to the dominates relationship that exists between elements in the lattice.

The distribution criteria routine of the RAM was augmented with additional code so that the shipping of fragments to relevant nodes based on the distribution criteria occurs automatically for each fragmentation scheme. We also adapted the CREATE table and ALTER table transactions to handle the attribute and element-level fragmentation schemes such that whenever a classification attribute or attribute value is shipped, the attribute or attribute value immediately preceding it in the schema must also be shipped along with it as part of the same fragment. Furthermore, we adapted the CQIM of the AUF facility to enforce a 'prohibited administrative operations' constraint on programmatic applications and interactive users such that only unprohibited DDL commands can be executed by users.

6.3.1 Capturing and Storing MLS information

The MLS information captured by the AUM includes the node name, node classification, user clearance, fragmentation scheme information, and the classification level of fragments. The node name and node classification information is captured at the point where the REGISTER node command is issued by the administrator. The user clearance information is captured when the administrator issues a CREATE user command; this information, like the previous ones is written to the CDB catalog via the CDM. Security classification labels are automatically inserted into a table's schema whenever the CREATE table command is executed. This additional attribute is also stored in the CDB catalog alongside other attributes of the table.

The MLS information stored in the CDB catalog is available to other modules in the Distributed server for use in processing. We are especially interested in how this information is used by the RAM module to only direct queries to authorised nodes. In the following sections we consider how this information is utilised in creating nodes, user accounts, tables, and in query processing.

6.3.2 Registering nodes in the cluster

Each node must be registered with the Distributed server before it can be considered a part of the cluster. The AUM handles node registrations for the Distributed server. To register a node, an administrator uses the REGISTER node command; this command has the form: REGISTER node <node name>[1] (<security classification>, <login password>). The Distributed server must know the security classification and the login passwords of all member nodes in order to communicate requests to them. The administrative request to register a node is compiled by the GDC module and forwarded directly to the DQEM module which then executes the command and writes the node information into the CDB via the CDM. This node classification information is also accessible to the RAM module. A node continues to exist in the CDB until it is dropped. The following example describes the command for registering a node called Inside with a classification of 'inside'.

- REGISTER node Inside (inside, shhh002);

A node that is registered with the Distributed server can be removed using the REMOVE node command; the command has the form REMOVE node <node name> (<login password>). When a node is removed from the cluster, its registration information in the CDB will no longer exist. The modified behaviour of the REMOVE node command guards against the possibility of 'dangling nodes' or 'dangling security labels'. There must exist a corresponding node for each security label that is in use; if a node is removed using the REMOVE node command, then its security label will be dropped and an exception error returned whenever the label is detected in a REGISTER node command or any other command that requires a security level parameter.

6.3.3 Creating user accounts

The CREATE user command is of the form: CREATE user <login name> (<clearance>, <password>). The user's security clearance was added to the parameter list to support MLS. The CREATE user request is compiled by the GDC module; this module also checks to see if the value entered for clearance is a valid lattice element. The request is forwarded to the DQEM module for execution. The profile of the new account is written to the CDB via the CDM. The user account profile, like the node registration information, also continues to exist until it is dropped. The RAM uses the security clearance information, stored in the user account profile, in combination with the node classification information to direct sub-queries to authorised nodes.

6.3.4 Creating databases

The CREATE database command provided by the COTS DBMS has the form: CREATEdb <database name> (<fragmentation option>). This command may only be executed by a user with administrative privileges. The CREATE database transaction is executed by the DQEM, which creates a distributed database, associates it with a specific fragmentation

[1]The process of registering a node relies on an earlier configuration procedure at the operating system level that associates a name assigned to a machine with the ethernet address of that machine.

scheme, and builds and populates a CDB catalog. The fragmentation option switch is used to specify a fragmentation scheme for tables of the newly created database; the only available options are the tuple, attribute, or element schemes. A fragmentation scheme specified by a CREATEdb statement persists until it is changed by a new CREATEdb statement specifying a different option. For each fragmentation scheme, the distribution criteria routine in the RAM module that administers its distribution criteria is activated during database creation. The command SETdf <database name> is used to set a database to the default database; it also resets the fragmentation scheme to that of the default database. The following example is a command for creating a database under the tuple-level fragmentation scheme.

- CREATEdb Hospital (tuple);

6.3.5 Creating tables

In contrast to the CREATE database command, new relations can be created by users logged in at System-low using the CREATE table DDL command; the request is then routed to the GDC module where it is compiled and forwarded to the DQEM module for execution. This command, along with the distribution criteria routine of the RAM, were adapted to no longer require the manual specification of the distribution criteria. The distribution criteria are now part of a routine that executes automatically to distribute fragments. Users cleared at any security level must *downgrade* their security clearance to System-low to be able to execute the CREATE table command. Relations are created and populated using the *insert low approach* described earlier in section 3.4.4. The CREATE table command of the COTS DBMS has the form: CREATE table <table name> (<attribute$_i$> <type> <size>, ..., <attribute$_n$> <type> <size>). The 'type' parameter defines the datatype of a field (e.g., CHARACTER, INTEGER, ALPHANUMERIC), and the 'size' parameter defines the size of a field (e.g., 5, 10). In processing the CREATE table request, the DQEM module generates an executable transaction that creates relation fragments based on the fragmentation scheme of the default database; the RAM is invoked afterwards to distribute the fragments based on the distribution criteria. Information on the newly created tables and their locations is written to the CDB via the CDM.

The following sections describe how tables are created, how they are stored, and how the Distributed server keeps track of them for each fragmentation scheme. The Department relation in Appendix A will be used as the basis of our discussions.

Tuple-level table creation

In this scheme, the TC attribute, its type, and size are automatically inserted into the schema whenever a new table is created, and the default type is LABEL. The tuple classification in the prototype is always named TC; TC is therefore reserved[2] and may not be used for naming other attributes. The following example is a command for creating a table under the tuple-level fragmentation scheme.

- **Command:** CREATE table Department (DN NUMBER(4), Dept_Name CHAR(10), Manpower CHAR(10), Dept_Location CHAR(10), Manager CHAR(10));
- **Modified command with distribution criteria inserted:** CREATE table Department (DN NUMBER(4) WITH
  (Dept_Name CHAR(25),
  Manpower CHAR(10),
  Dept_Location CHAR(25),
  Manager CHAR(25),
  TC LABEL(20)),
  @ High Node IF TC = 'System-high',
  @ Outside Node IF TC = 'outside',
  @ Inside Node IF TC = 'inside',
  @ Low Node IF TC = 'System-low');

Attribute-level table creation

In this scheme, the classification attributes denoted by $Class\_1, \ldots, Class\_n$, their types, and sizes are automatically inserted into the schema whenever a new table is created; the default type is LABEL. Any attribute name of the form: '$Class\_i$', where i is any integer value, is reserved in the system for the naming of attribute labels. The first tuple to be inserted must contain values in all of its classification fields; this tuple must be inserted by the administrator logged in at System-low, because it defines the classification level of all the attribute fragments. Tuples inserted afterwards by users are not required to have values in these fields. The distribution criteria routine scans only the classification fields of the first row to determine the destination of fragments; it does not scan subsequent entries. An attribute fragment is automatically shipped to a relevant node based on the distribution criteria specified on the classification attributes in the distribution criteria routine. This routine uses the node names captured by the AUM during node registration in its processing. The following example describes the command used to create a table for the attribute-level fragmentation scheme.

- **Command:** CREATE table Department (DN NUMBER(4), Dept_Name CHAR(10), Manpower CHAR(10), Dept_Location CHAR(10), Manager CHAR(10));
- **Modified command with distribution criteria inserted:** CREATE table Department (DN NUMBER(4), $Class\_1$ LABEL(20), WITH

[2]Reserved words are maintained in an identifier file called *idf.sys*; this file includes both user and Stargres reserved words.

```
(Dept_Name CHAR(25), $Class$_2 LABEL(20),
@ High Node IF $Class$_2 = ‘System-high’,
@ Outside Node IF $Class$_2 = ‘outside’,
@ Inside Node IF $Class$_2 = ‘inside’,
@ Low Node IF $Class$_2 = ‘System-low’;

Manpower CHAR(10), $Class$_3 LABEL(20),
@ High Node IF $Class$_3 = ‘System-high’,
@ Outside Node IF $Class$_3 = ‘outside’,
@ Inside Node IF $Class$_3 = ‘inside’,
@ Low Node IF $Class$_3 = ‘System-low’;

Dept_Location CHAR(25), $Class$_4 LABEL(20),
@ High Node IF $Class$_4 = ‘System-high’,
@ Outside Node IF $Class$_4 = ‘outside’,
@ Inside Node IF $Class$_4 = ‘inside’,
@ Low Node IF $Class$_4 = ‘System-low’;

Manager CHAR(25), $Class$_5 LABEL (20),
@ High Node IF $Class$_5 = ‘System-high’,
@ Outside Node IF $Class$_5 = ‘outside’,
@ Inside Node IF $Class$_5 = ‘inside’,
@ Low Node IF $Class$_5 = ‘System-low’));
```

Element-level table creation

In this scheme, the classification attributes denoted by the reserved words $Class$_1, ..., $Class$_n, their types, and sizes are automatically inserted into the schema whenever a new table is created; the default type is LABEL. Like the attribute-level scheme, any attribute name of the form: ‘$Class$_i’, where i is any integer value, is reserved for the naming of attribute labels. The distribution criteria are specified on the classification attribute values $Class$_1, ..., $Class$_n of each element in the tuple. When new tuples are added to the table, each element fragment will be examined automatically by the distribution criteria routine and shipped to the relevant node based on the distribution criteria. The process of creating tables under this fragmentation scheme is similar to that of the attribute-level scheme, except that the classification values may vary, and the distribution criteria routine must scan the classification values in every row to determine the destination of fragments. The following example describes the command used to create a table for the element-level fragmentation scheme.

- **Command:** CREATE table Department (DN NUMBER(4), Dept_Name CHAR(10), Manpower CHAR(10), Dept_Location CHAR(10), Manager CHAR(10));
- **Modified command with distribution criteria inserted:** CREATE table Department (DN NUMBER(4), $Class$_1 LABEL(20), WITH
  (Dept_Name CHAR(25), $Class$_2 LABEL(20),
  @ High Node IF $Class$_2 = 'System-high',
  @ Outside Node IF $Class$_2 = 'outside',
  @ Inside Node IF $Class$_2 = 'inside',
  @ Low Node IF $Class$_2 = 'System-low';

  Manpower CHAR(10), $Class$_3 LABEL(20),
  @ High Node IF $Class$_3 = 'System-high',
  @ Outside Node IF $Class$_3 = 'outside',
  @ Inside Node IF $Class$_3 = 'inside',
  @ Low Node IF $Class$_3 = 'System-low';

  Dept_Location CHAR(25), $Class$_4 LABEL(20),
  @ High Node IF $Class$_4 = 'System-high',
  @ Outside Node IF $Class$_4 = 'outside',
  @ Inside Node IF $Class$_4 = 'inside',
  @ Low Node IF $Class$_4 = 'System-low';

  Manager CHAR(25), $Class$_5 LABEL (20),
  @ High Node IF $Class$_5 = 'System-high',
  @ Outside Node IF $Class$_5 = 'outside',
  @ Inside Node IF $Class$_5 = 'inside',
  @ Low Node IF $Class$_5 = 'System-low'));

Keeping track of fragments

The Distributed server keeps track of the status and location of fragments through the registration information maintained in the CDB catalog. This information, captured during the CREATE table operation, was described earlier in this section. More specifically, it keeps track of the tuple fragments by associating each tuple with its parent relation in the catalog, or, for attribute and element fragments, by associating each fragment with its primary key and parent relation in the catalog. The relation schema of the parent relation is necessary to unite all the disjoint fragments of the same table.

The distribution of fragments to relevant nodes is dependent on the relevant nodes being accessible. A node may become inaccessible because it has been removed from the cluster using the REMOVE node command, or for some other reason. Fragments cannot be shipped to a node that has been 'legally' removed using the REMOVE node command, as its location and registration information is no longer available for routing purposes; the distribution criteria routine will not reference any node that is removed this way. If a node is inaccessible for some other reason, the Distributed server will attempt to contact the node, resulting in an error message. In this scenario, the shipping of all fragments of the same table is treated as one atomic transaction; it must either succeed to all the nodes or fail, thereby preserving the completeness property of a table.

6.3.6 Altering the tables

The ALTER TABLE commands work by constructing a temporary copy of the original table from the single-level fragments. The addition or alteration is performed on the copy; the copy is then fragmented, the original fragments are deleted, and the new ones are renamed as the originals. This is done in such a way that all updates are automatically redirected to the new fragments without any failed updates. While ALTER TABLE is executing, the original table is readable by other clients. Updates and writes to the original fragments are stalled until the new fragments are ready.

To add a column in a tuple-level fragmentation scheme, the necessary command has the form ALTER TABLE <table name> ADD (<attribute$_i$> <type> <size>). For the attribute and element-level fragmentation schemes, a classification attribute is automatically added for each column that is affixed. We adapted the command to the form: ALTER TABLE <table name> ADD (<attribute$_i$> <type> <size>, <Class$_i$> <type> <size>). Its effect is to create two columns; the first column takes the name specified by the user, and the second column, inserted automatically, is named 'Class$_i$' by default. To alter a table by modifying the properties of an existing column, the generic command has the form ALTER TABLE <table name> MODIFY (<attribute$_i$> <type> <size>). The global schema in the CDB catalog is updated after the command is executed by the DQEM module.

6.3.7 Query processing

Each request for data is forwarded by the Distributed server to the appropriate nodes containing the referenced fragments. This process begins when a query is received by the CQIM in the AUF facility. The query is filtered by a filter function in the CQIM to prevent administrative operations, such as CREATE database, REGISTER node, or REMOVE node, from being executed by non-administrative users. The filter function, called *Sievadmincmd*, checks the commands in the query against a list of these prohibited administrative commands; any command that is present in the list is rejected and an error message is returned. Afterwards, if the query is considered legal, it is passed through the Distributed server's query processing pipeline.

In the pipeline, table references are verified by the DQM module, the query is transformed, optimised, an execution strategy is developed, and the RAM is invoked to distribute the sub-queries to local sites for further processing. The RAM uses the node classification and user clearance information maintained in the CDB catalog when deciding which nodes to contact. Queries to unauthorised nodes are silently dropped, and the target nodes are never contacted. Sub-queries destined for other nodes in the cluster are forwarded to the NSV module of the TMF facility. The NSV module opens a connection to the NSV modules of the nodes authorised by the RAM. Recall that if the RAM chooses not to forward the password of a remote node as part of its argument to the NSV module, the NSV module will be unable to establish a connection. This feature places the NSV's outbound traffic under the direction of the RAM. The RAM uses the node name, node address, and table location information maintained in the CDB catalog to determine which nodes to contact; every fragment is associated with a specific node in a manner that allows the DBMS to determine the nodes to be contacted for each request that it receives. A diagramatic representation of the main processing steps for an arbitrary DML query is illustrated in figure 6.1.

At each authorised local node, the query is passed through the pipeline of the local query processor where it is parsed, preprocessed, subjected to integrity constraint verification, and finally optimized before reaching the query execution engine. At the execution engine, the query is executed locally using local resources and the partial result is forwarded to the local NSV module which transmits it to the Distributed server node. On receipt of incoming partial results, the NSV module running at the Distributed server node forwards it to the DQM module for further processing. The DQM module combines the partial results using Union operations for the tuple-level fragmentation scheme, Join operations for the attribute-level fragmentation scheme, and a combination of Union and Join operations for the element-level fragmentation scheme. The full result set is returned to the application program or interactive user.

As a result of time and effort constraints, only the Select, Join, Insert, Update, and Delete DML commands were adapted in the MST prototype. An error message will be returned by the prototype if it encounters a DML query other than one of these five. The following examples show how Select, Insert, and Update queries are processed for each fragmentation scheme. The Department relation in Appendix A will be used as the basis of our discussions.

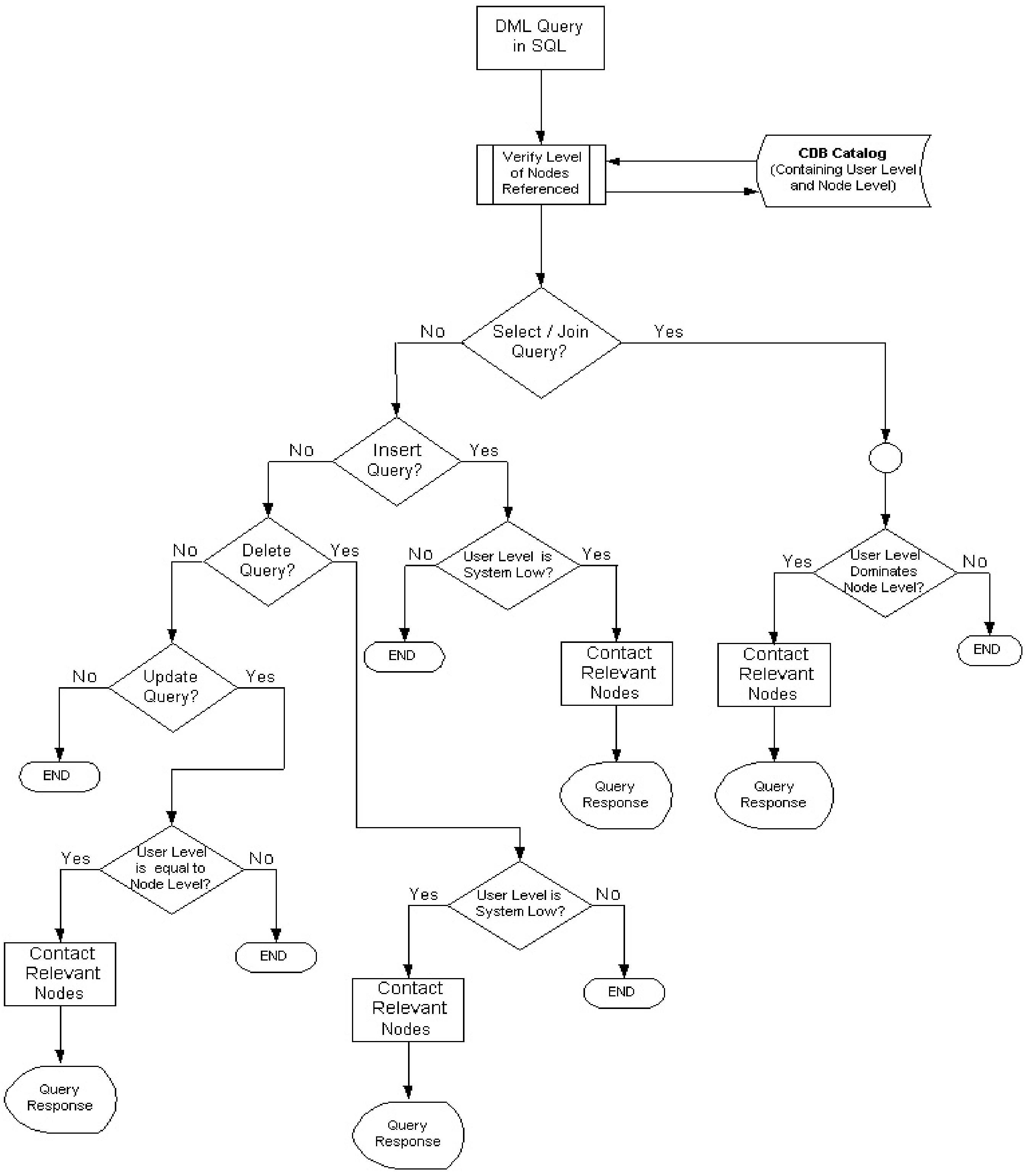
DML Query in SQL
Verify Level of Nodes Referenced
CDB Catalog (Containing User Level and Node Level)
Select / Join Query?
No
Yes
Insert Query?
No
Yes
Delete Query?
No
Yes
User Level is System Low?
No
Yes
User Level Dominates Node Level?
Yes
No
END
Contact Relevant Nodes
Query Response
Update Query?
No
Yes
User Level is equal to Node Level?
Yes
No
User Level is System Low?
Yes
No

Figure 6.1: A flowchart illustration of how the Distributed server handles an arbitrary DML query.

Tuple-level fragmentation scheme

Select queries and Join queries are handled in much the same way; the user's clearance must dominate the classification of a fragment referenced in order for its host node to be contacted. For every sub-query that reaches the RAM, the user's clearance is compared with the classification of the target node. A connection request is denied to a node if the user's clearance does not dominate it, and the target nodes are never contacted in this case. A query is only forwarded to nodes dominated by the user's clearance. The following example illustrates how a query is handled for a user with security clearance 'outside'.

Example

- **User Query:** Select ⋆ FROM Department
- **Modified query with distribution criteria inserted:** Select DN, Dept_Name, Manpower, Dept_Location, Manager, TC FROM Department
  @ Department.Hospital:Node[Outside, Low]

For the Insert and Update SQL queries, the user's clearance must be at System-low (for Insert) and equal to the node classification (for Update) in order to execute these commands. As the processing of the Insert and Update requests also progresses in much the same way, the following example illustrates the Insert query for a user with security clearance 'System-low'.

Example

- **User Query:** Insert INTO Department Values (D30, Psychiatry, 8, 64 South Wing, Fraser, System-high);
- **Modified query with distribution criteria inserted:** Insert INTO Department Values (D30, Psychiatry, 8, 64 South Wing, Fraser, System-high)
  @ Department.Hospital:Node[High]
- **Result:** Insert Successful. Tuple inserted @ Node[High]

Attribute-level fragmentation scheme

In this scheme, the Select queries and Join queries are also handled in much the same way. The user's clearance must dominate the classification of each node containing fragments referenced before it can be contacted. Whenever a sub-query reaches the RAM, the user's clearance is compared with the classification of the target node, and a connection request is denied if the user's clearance does not dominate a target node. A node is only contacted, and a query forwarded to it, if it is dominated by the user's clearance. The following example illustrates how a query is handled for a user with security clearance 'outside'.

Example

- **User Query:** Select $\star$ FROM Department
- **Modified query with distribution criteria inserted:** Select DN, $Class\_1$, Dept_Name, $Class\_2$, Manpower, $Class\_3$, Dept_Location, $Class\_4$, Manager, $Class\_5$ FROM Department @ Department.Hospital:Node[Outside, Low]

For the Insert and Update SQL queries, the user's clearance must be at System-low (for Insert) and equal to the node classification (for Update) in order to execute these commands. The next example illustrates the Insert query, as it is handled in a similar way to the Update query. Recall from section 6.3.5 that the first tuple that defines the classification of each attribute fragment must be inserted by the administrator logged in at System-low. Example 2 illustrates the Insert query for a user with security clearance System-low.

Example: 1

- **Administrator Query:** Insert INTO Department Values (D30, System-low, Psychiatry, System-low, 8, inside, 64 South Wing, outside, Fraser, System-high);
- **Modified query with distribution criteria inserted:** Insert INTO Department Values (D30, System-low, Psychiatry, System-low, 8, inside, 64 South Wing, outside, Fraser, System-high) @ Department.Hospital:Node[High, Outside, Inside, Low]
- **Result:** Insert Successful. Tuple inserted @ Node[High, Outside, Inside, Low]

Example: 2

- **User Query:** Insert INTO Department Values (D30, Psychiatry, 8, 64 South Wing, Fraser);
- **Modified query with distribution criteria inserted:** Insert INTO Department Values (D30, System-low, Psychiatry, System-low, 8, inside, 64 South Wing, outside, Fraser, System-high) @ Department.Hospital:Node[High, Outside, Inside, Low]
- **Result:** Insert Successful. Tuple inserted @ Node[High, Outside, Inside, Low]

Element-level fragmentation scheme

This scheme is very similar to the attribute-level scheme described earlier; the Select queries and Join queries are also handled in much the same way, and the user's clearance must dominate the classification of the node containing referenced fragments before it can be contacted. When a sub-query reaches the RAM, the user's clearance is compared with the classification of the target nodes, and connection requests are granted only to nodes dominated by the user's clearance; nodes not dominated by the user's clearance are never contacted. The following example illustrates how a query is handled for a user with security clearance 'outside'.

Example

- **User Query:** Select $\star$ FROM Department
- **Modified query with distribution criteria inserted:** Select DN, $Class$_1, Dept_Name, $Class$_2, Manpower, $Class$_3, Dept_Location, $Class$_4, Manager, $Class$_5 FROM Department @ Department.Hospital:Node[Outside, Low]

For the Insert and Update SQL queries, the user's clearance must be at System-low (for Insert) and equal to the node classification (for Update) in order to execute these commands. The following example illustrates the Insert query for a user with security clearance System-low; it is handled in a similar way to an Update query.

Example

- **User Query:** Insert INTO Department Values (D30, System-low, Psychiatry, System-low, 8, inside, 64 South Wing, outside, Fraser, System-high);
- **Modified query with distribution criteria inserted:** Insert INTO Department Values (D30, System-low, Psychiatry, System-low, 8, inside, 64 South Wing, outside, Fraser, System-high) @ Department.Hospital:Node[High, Outside, Inside, Low]
- **Result:** Insert Successful. Tuple inserted @ Node[High, Outside, Inside, Low]

6.4 Summary

In this chapter we have presented a technical description of how a COTS DBMS product was adapted to provide multilevel security. Our adaptation efforts focused on the two main subsystems of Stargres, namely the DBMS and the Distributed server. Although our design focused on the adaptation of the Distributed server to capture, store, and use MLS information in creating user accounts, creating tables, registering nodes, and distributing queries, a number of modules provided by the DBMS component were also essential to the design. Especially important were the AUM and CQIM modules of the AUF facility and the NSV module of the TMF facility which, although external to the Distributed server, are indispensable to a robust prototype.

Programmatic and interactive DML queries are screened for prohibited administrative operations at the AUF facility. Overall, the prototype relies on the Distributed server component to prevent access to unauthorised fragments and to perform all the other distributed tasks such as query optimisation, query decomposition, site selection, and the coordination required between local DBMSs. However, the Distributed server still relies on the local DBMSs to perform local optimisations and query execution on its behalf. The order in which these operations are executed is entirely controlled by the Distributed server in liaison with the local DBMSs.

Chapter 7 investigates the performance of the three fragmentation schemes supported by the prototype to determine the impact of varying the size and structure of the database on the performance of the front-end and network.

Chapter 7

Performance Study of Fragmentation Schemes

7.1 Introduction

This chapter presents a performance study of the tuple, attribute, and element-level fragmentation schemes of the MST prototype. An experimental database, the *Hospital database* comprising eight relations was created and populated to facilitate the study. The fragments of the relations are distributed to the various nodes in the network according to their security levels; this ensures that fragments can only be maintained at nodes with corresponding security levels. Any number of nodes can be utilised, but the numbers must match the set of labels in the security lattice of the database without redundancy.

In the design of our experiments to determine the relative performance of the three fragmentation schemes, we identified two areas of investigation, namely the front-end and the back-end. These areas are considered to be sufficiently sensitive to the effects of performance inhibitors as to provide us with an adequate measure of their impact. The performance of a distributed database management system depends on a number of different factors; for example, if the DBMS references data over a database link, the performance of the network infrastructure will have a direct bearing on the overall performance of the system. In addition, activities such as snapshot refreshes and data propagation also impact performance, and, to a large extent, their impact is related to the size of the database. At the front-end, performance is affected by such things as user authentication and the post-processing of query responses. The page size is one of the many factors that affect the performance of the back-end DBMS. This work builds on a previous performance study described in [77].

The following section describes the environment and organisation of the experiments. Section 7.3 describes the structure of the database that was used in our experiments; this structure is modified to fit the requirements of each fragmentation scheme. Section 7.4

identifies specific performance inhibitors that impact the performance of MLS/DBMS. Sections 7.5 and 7.6 present the experiments conducted and their results for the Select-All and Join queries at all three levels of fragmentation. Section 7.7 presents the experiments conducted and their results for Update operations at all three levels of fragmentation. Our analysis of the results from all the experiments is presented in section 7.8. We conclude with a summary of our observations in section 7.9.

7.2 Organisation of the experiments

The local area network that is the backbone of our implementation consists of sixteen 933MHz Pentium III machines. Each machine is fitted with a 100 Megabit network adapter card, 512MB of primary memory, 256K of cache memory, and a 36GB SCSI hard disk. This configuration was determined in part by the operational requirements of the DBMS. For example, Stargres requires 512MB of primary memory in order to minimize paging and operate optimally, and SCSI hard drives allow for multiple concurrent read/writes which is ideally suited to I/O bound applications like DBMSs. Our network configuration was also constrained by the number of machines and other hardware devices that we had available.

The examples in chapter 6 were illustrated using a lattice of 4 partially ordered security labels. The experiments described in this chapter will utilise a lattice of 16 totally ordered security labels (1, 2, ..., 16). The experiments use the CPU response time (in seconds) as metric. The measurements were captured at the Distributed server using a Visual DBA monitoring tool provided by Stargres. For each query, the tool records the time it takes for the system to respond to a query. In addition, we also utilise a Linux network monitoring utility to monitor network traffic at the front-end.

For each experiment, we plot the response times in a graph as a function of the variable whose impact is being investigated. All three fragmentation schemes are also shown on the same graph for ease of comparison. Furthermore, for each experiment, the set of response times is computed by taking the average response times of users at all security levels in 16 repeat experiments. There are in total, 1680 tuples and 48 attributes between the relations in the Hospital database. For each fragmentation scheme, the number of fragments are equally distributed between 16 security levels in each experiment.

7.3 Experimental database structure

The Hospital database consists of eight inter-related tables of varying sizes. We chose a database size that was sufficiently large to provide us with meaningful measurements, but we are mindful of the fact that its size may have to be increased if our experiments show that it does not sufficiently stress the system. In interrogating the database, in order to ensure that the database is reasonably stressed we concentrated on two queries for processing relation fragments: the Select-All query which selects all the tuples in a relation, and the Join query which joins two relations. These two queries are amongst the most expensive operations in query processing and have been used by other researchers conducting performance studies [77]. The following schema represents the structure of the relations in the database. A subset of the *Patient*, *Department*, and *Nurse* relations is shown in Appendix A.

The patient table

Patient(P_SSN, P_Name, P_DOB, P_Status, P_Tel., INS_Name, PHY_Name, EC_Name, Patient #)

The Patient table provides information about patients who sought admission to the hospital; it has 9 attributes and 512 tuples. *P_SSN* is the primary key of this relation. We will refer to this relation as R_1 in our experiments and discussions.

The emergency contact (EC) table

EC(EC_SSN, EC_Name, EC_Tel., Relationship, State)

The Emergency Contact (EC) table provides information about a relative of each patient that may be contacted in an emergency. The table has 5 attributes and 512 tuples, each entry corresponds to an entry in the *Patient* relation. *EC_SSN* is the primary key of this relation. We will refer to this relation as R_2 in our experiments and discussions.

The insurance table

Insurance(INS_Name, INS_Location, INS_Tel., INS_Fax, INS_Contact)

The Insurance table contains information on the patient's insurance provider; it has 5 attributes and 16 tuples. *INS_SSN* is the primary key of this relation. We will refer to this relation as R_3 in our experiments and discussions.

The department table

Department(DN, Dept_Name, Manpower, Dept_Location, Manager)

The Department table contains information about each department at the hospital; it has 5 attributes and 16 tuples. *DN* is the primary key of this relation. We will refer to this relation as R_4 in our experiments and discussions.

The physician table

Physician(PHY_SSN, PHY_Name, PHY_DOB, PHY_DOR, PHY_Salary, PHY_Specialty, DN, PHY_Tel., PHY_Pager)

The Physician table provides information about each physician at the hospital; it has 9 attributes and 32 tuples. *PHY_SSN* is the primary key of this relation. We will refer to this relation as R_5 in our experiments and discussions.

The nurse table

Nurse(N_SSN, N_Name, N_DOB, N_DOR, N_Salary, N_Specialty, N_Pager, DN, Manager)

The Nurse table provides information about each nurse at the hospital; it has 9 attributes and 64 tuples. *N_SSN* is the primary key of this relation. We will refer to this relation as R_6 in our experiments and discussions.

The encounter table

Encounter(Patient #, Symptoms, Diagnosis, Condition, Date, Time, PHY_Name, N_Name, Action)

The Encounter table provides information about each patient encounter with the hospital; it has 9 attributes and 512 tuples. *Patient #* is the primary key of this relation. We will refer to this relation as R_7 in our experiments and discussions.

The facilities table

Facilities(Facility, Location, Function, Status, Caretaker)

The Facilities table provides information about each surgical facility at the hospital, it has 5 attributes and 16 tuples. *Facility* is the primary key of this relation. We will refer to this relation as R_8 in our experiments and discussions.

7.4 Identifying the inhibitors and investigating their effects

The main inhibitors that affect the performance of multilevel DBMSs include the size of the member relations (based on the number of tuples and attributes), the number of security levels in the lattice, the page size, and the architecture of the DBMS. We also recognise that things such as the Selectivity factor (how many result tuples or attributes are produced by the query), or Join factor (how many source tuples, attributes, or elements are being joined) also affect performance. In the investigations that follow, we identify some specific performance inhibitors, and then design experiments to investigate their impact on the front-end and back-end of the three fragmentation schemes using Select-All and Join queries. The quantity of each inhibitor (or variable) will be varied a number of times to study its effect on the response times.

7.5 Select-All query

The following experiments investigated the impact of varying the number of tuples, number of attributes, number of security levels, user's clearance level, and the page size on the performance of the front-end and back-end of the three fragmentation schemes.

7.5.1 Impact of varying the number of tuples

This experiment was designed to determine if the cost of processing varying numbers of tuples and shipping them across the network has a significant impact on the performance of the front-end and back-end of each fragmentation scheme. We chose this experiment because the size of a database is, in part, based on the number of its tuples (records), and the number of tuples processed during each transaction could determine how long it takes to return a response to a user. In addition, when the tuples are disjoint and distributed, more time is expended by queries that must combine these disjoint fragments.

Operational conditions for figure 7.1

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_6
Original Query: Select-All FROM R_6
Criteria: Vary number of tuples to 112, 212, 312, 412, and 512; fix # of attributes at 9; fix # of security levels at 16; fix user's clearance level; fix page size at 8.
X axis: # of tuples in relation R_6.
Y axis: CPU time in seconds.

Observation and explanation of results

The response times exhibited linear growth as the number of tuples was increased (see figure 7.1 for the Select-All query results and figure 7.6 for the Join query results). The observed linear growth is an indication of the increased number of Unions (for the tuple-level fragmentation scheme), Joins (for the attribute-level fragmentation scheme), Unions and Joins (for the element-level fragmentation scheme), that were needed in reconstructing the response. Although the response times observed in the experiments measured the

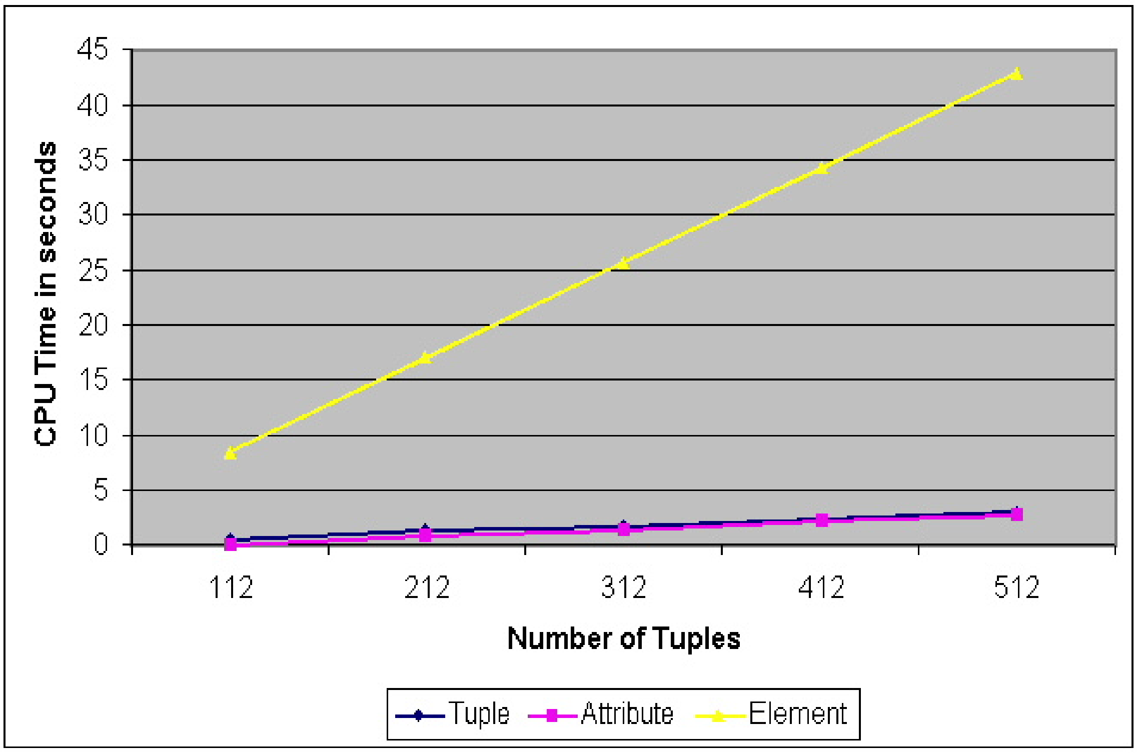

Figure 7.1: Impact of varying the number of tuples to 112, 212, 312, 412, and 512; fix # of attributes at 9; fix # of security levels at 16; fix user's clearance level; fix page size at 8. (Select-All Query).

degradation at the front-end, it does not show how this degradation is affected by other inhibitors at the back-end. We will investigate this question in the following experiment.

7.5.2 Impact of page size on the back-end processing costs in section 7.5.1

This experiment investigates the impact of the operating system page size on the back-end processing costs. We designed this experiment in order to further investigate how other less noticeable performance inhibitors at the back-end contribute to the overall processing cost.

Operational conditions for figure 7.2

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_6
Original Query: Select-All FROM R_6
Criteria: Vary the page size to 2, 4, 6, and 8. Fix # of tuples in each node at 105; fix # of attributes at 9; fix # of security levels at 16; fix user's clearance level.
X axis: Operating system page size (in Megabytes).
Y axis: CPU time in seconds.

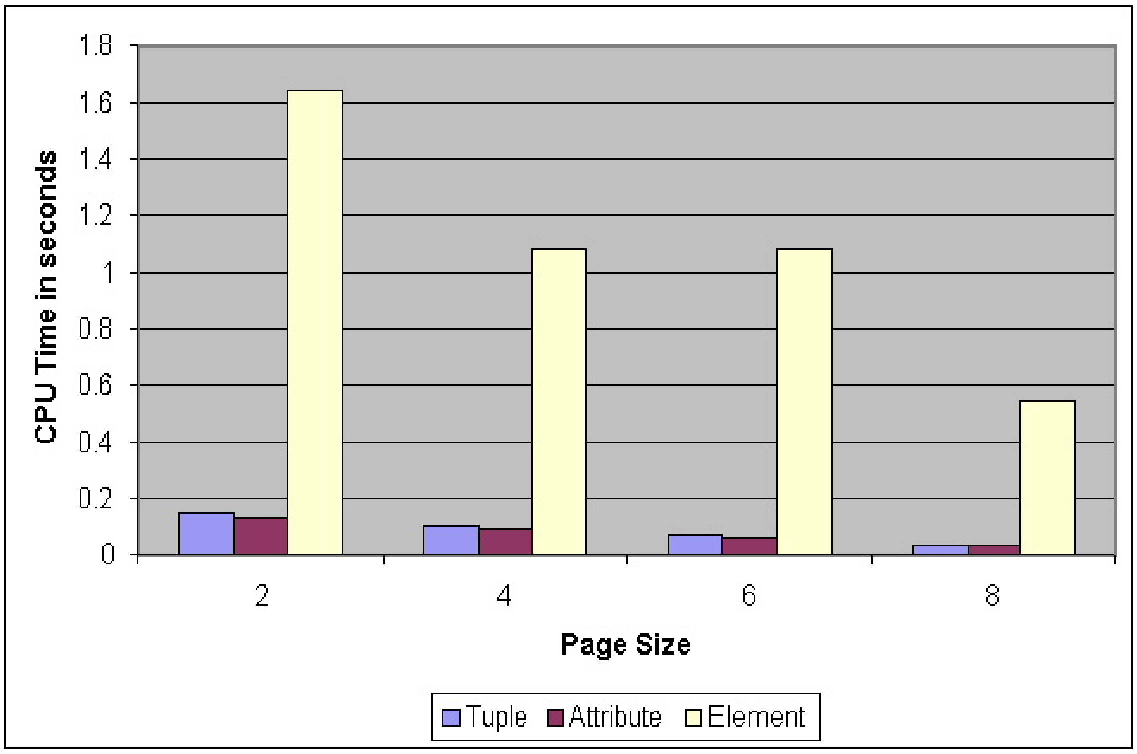

Figure 7.2: Vary the page size to 2, 4, 6, and 8. Fix # of tuples in each node at 105; fix # of attributes at 9; fix # of security levels at 16; fix user's clearance level. (Select-All Query).

Observation and explanation of results

The default page size is 8MB. The response times increased significantly for all three fragmentation schemes as the page size was reduced, (see figures 7.2 and 7.7). A reduction in page size means that the unit of transfer between disk and memory (or the physical record size) is reduced, thereby resulting in increased paging activity as the operating system swaps these units in and out of the DBMS buffers in main memory.

7.5.3 Impact of varying the number of attributes

This experiment investigates the impact of another primary inhibitor, the attribute, which may provide us with further information about the relative impact of all the factors examined so far. The rationale behind this inquiry is that, as the size of a database is a function of its tuples and attribute numbers, it is reasonable to try and find out the relative weighting of these variables on performance.

Operational conditions for figure 7.3

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_6

Original Query: Select-All FROM R_6
Criteria: Vary number of attributes to 2, 4, 6, and 8; fix # of tuples at 512; fix # of security levels at 16; fix user's clearance level; fix page size at 8.
X axis: # of attributes in relation R_6.
Y axis: CPU time in seconds.

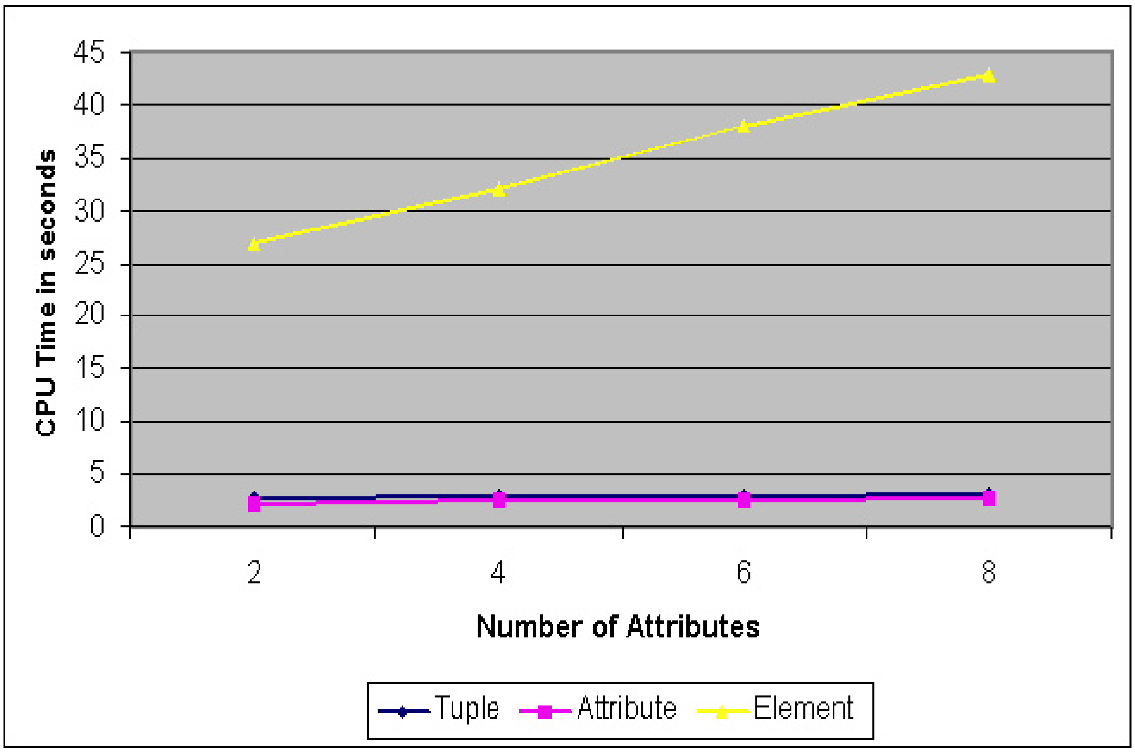

Figure 7.3: Vary number of attributes to 2, 4, 6, and 8; fix # of tuples at 512; fix # of security levels at 16; fix user's clearance level; fix page size at 8. (Select-All Query).

Observation and explanation of results

The response times for all three fragmentation schemes exhibited linear growth as the number of attributes was increased, (see figures 7.3 and 7.8). As the number of attributes increase, so also does the number of fragments that must be combined by Union and/or Join operations to provide a response, and hence the growth in response times. The relative impact of attributes on performance compared to, say, the number of security levels is still undetermined. The next experiment investigates this question.

7.5.4 Impact of varying the number of security levels

We designed this experiment to investigate the impact of varying the number of security levels on the performance of the three fragmentation schemes. The impact of security levels is an appealing line of investigation because the distribution of relation fragments is based on security level, and how these fragments are distributed in turn determines the cost of the recovery algorithm that combines the fragments.

Operational conditions for figure 7.4

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_6
Original Query: Select-All FROM R_6
Criteria: Vary number of security levels to 2, 4, 6, 8, 10, 12, 14, and 16; fix # of tuples at 512; fix # of attributes at 9; fix user's clearance level; fix page size at 8.
X axis: # of security levels in relation R_6.
Y axis: CPU time in seconds.

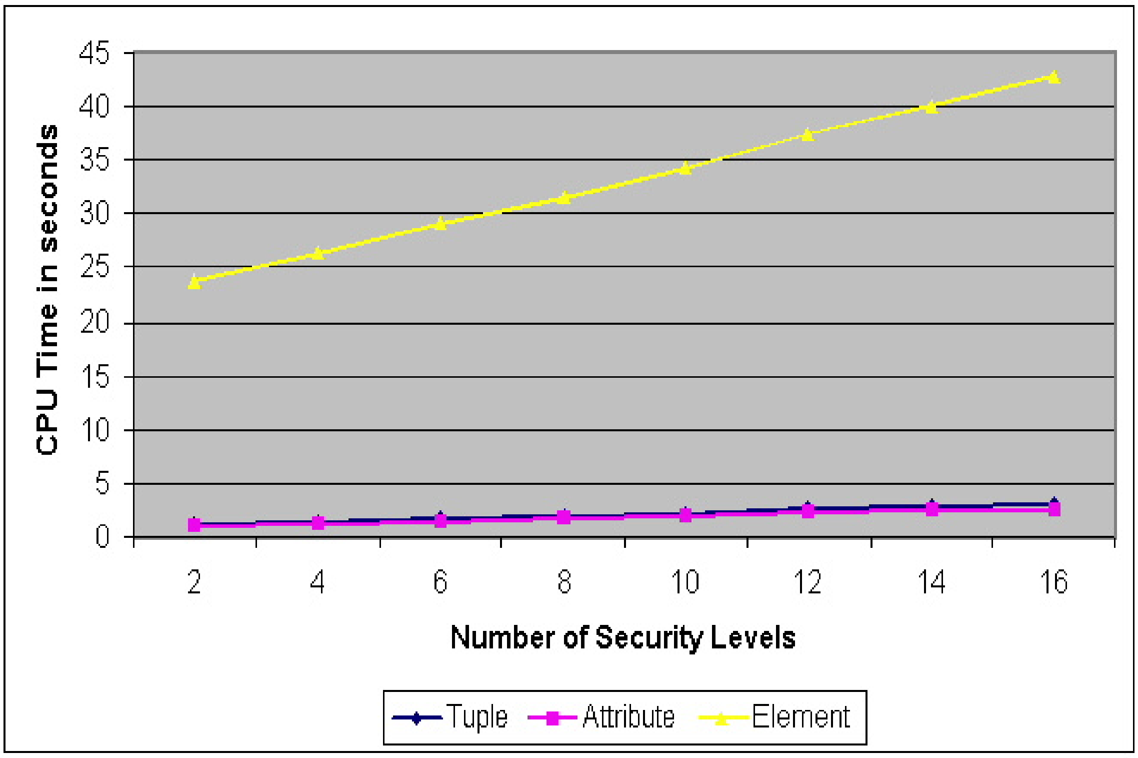

Figure 7.4: Vary number of security levels to 2, 4, 6, 8, 10, 12, 14, and 16; fix # of tuples at 512; fix # of attributes at 9; fix user's clearance level; fix page size at 8. (Select-All Query).

Observation and explanation of results

The response times exhibited linear growth as the number of security levels was increased (see figures 7.4 and 7.9). The result shows that, as the number of security levels are decreased, there is a consequent effect on how many relation fragments are created, and hence recovered. As indicated earlier in section 7.5.1, an increase in the number of fragments to be recovered translates to an increase in the number of Union and/or Join operations that must be performed in order to provide a response. A multilevel relation with 1 security level is not dissimilar to a non-multilevel relation.

7.5.5 Impact of changing the number of users on CPU Load

This experiment was designed to investigate and compare the overall impact on CPU Load when a single user submits a query as opposed to queries from 16 concurrent users and the CPU Load on a single node. More specifically, we wish to determine if the CPU Load on nodes with higher classifications is greater or smaller than the CPU Load on nodes with lower classifications.

Operational conditions for figure 7.5

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_6
Original Query: Select-All FROM R_6
Criteria: Vary # of concurrent users. Fix # of tuples at 512; fix # of attributes at 9; fix user's clearance level; fix # of security levels at 16; fix page size at 8.
X axis: # of concurrent users.
Y axis: CPU time in seconds.

Observation and explanation of results

Our observation revealed that the CPU Load increased as the number of concurrent users were increased. The load distribution appears to be evenly spread between the different nodes in the network for a single user at level-16, but increases significantly when the number of users increases to 8 (see figures 7.5 and 7.10). The higher level nodes appear to have less load compared to lower level nodes. This increase may be attributed to the fact that lower level nodes are most frequently accessed because of their security level, as opposed to higher level nodes which are only referenced by fewer user queries.

7.6 Join query

The experiments presented in this section use a Join query to explore the performance of the MST prototype's three fragmentation schemes. We plot the response times for varying the number of tuples, the number of attributes, the number of security levels, the user's clearance level, and the page size. The Join operation now involves relations R_1, R_2, R_3,

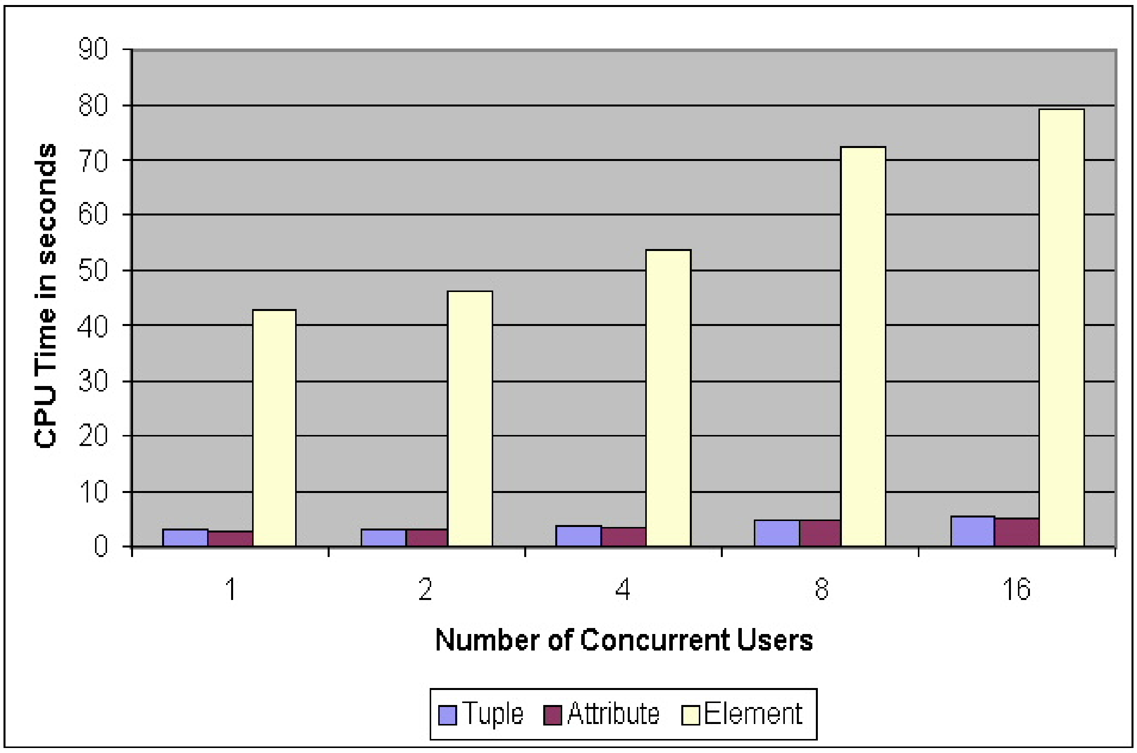

Figure 7.5: Vary # of concurrent users. Fix # of tuples at 512; fix # of attributes at 9; fix user's clearance level; fix # of security levels at 16; fix page size at 8. (Select-All Query).

R_4, R_5, R_6, R_7, and R_8; as a result, the number of tuples has increased to 1680, and the number of attributes has increased to 48. The number of security levels and the user clearances remain unchanged.

7.6.1 Impact of varying the number of tuples

See section 7.5.1 for an explanation of this experiment and the results.

Operational conditions for figure 7.6

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_1, R_2, R_3, R_4, R_5, R_6, R_7, and R_8
Original Query: Join $R_1 \ldots R_8$
Criteria: Vary number of input tuples to 380, 680, 980, 1280, and 1680; fix # of attributes at 48; fix # of security levels at 16; fix user's clearance level; fix page size at 8.
X axis: # of tuples in all relations.
Y axis: CPU time in seconds.

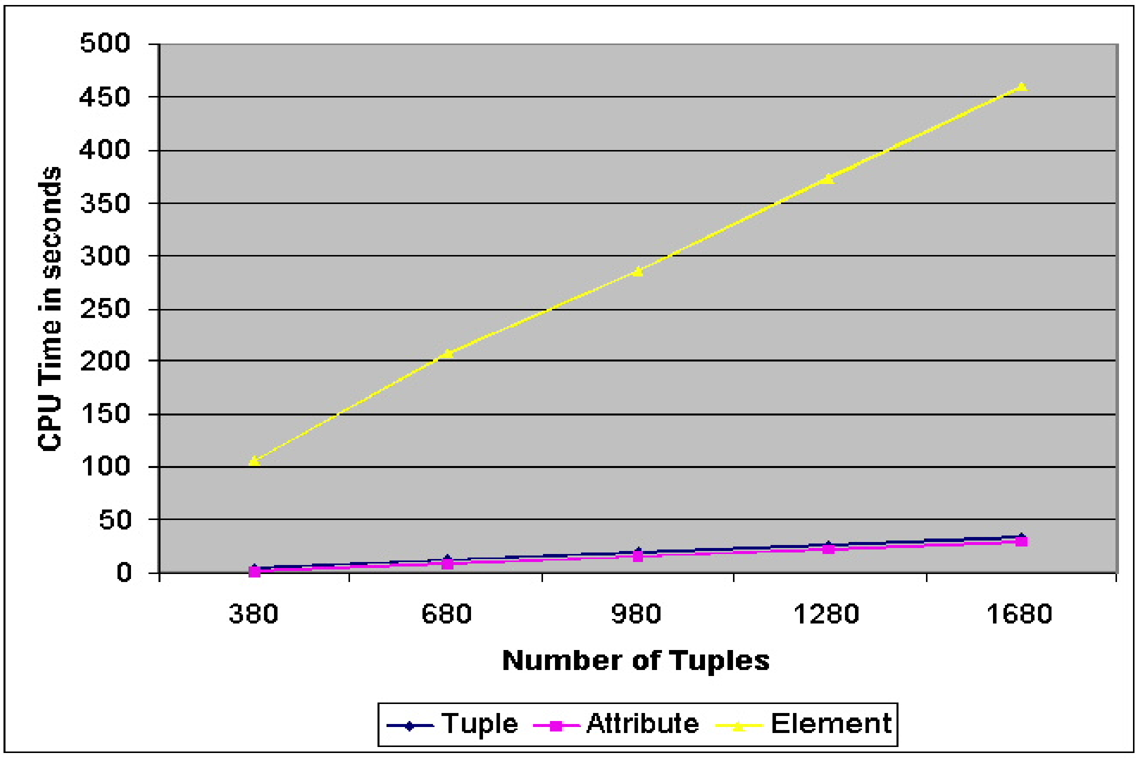

Figure 7.6: Impact of varying the number of tuples to 380, 680, 980, 1280, and 1680. Fix # of attributes at 48; fix # of security levels at 16; fix user's clearance level; fix page size at 8. (Join Query).

7.6.2 Impact of page size on the back-end processing costs in section 7.6.1

See section 7.5.2 for an explanation of this experiment and the results.

Operational conditions for figure 7.7

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_1, R_2, R_3, R_4, R_5, R_6, R_7, and R_8
Original Query: Join $R_1 \ldots R_8$
Criteria: Vary the page size to 2, 4, 6, and 8. Fix # of tuples in each node at 105; fix # of attributes at 48; fix # of security levels at 16; fix user's clearance level.
X axis: Operating system page size (in Megabytes).
Y axis: CPU time in seconds.

7.6.3 Impact of varying the number of attributes

See section 7.5.3 for an explanation of this experiment and the results.

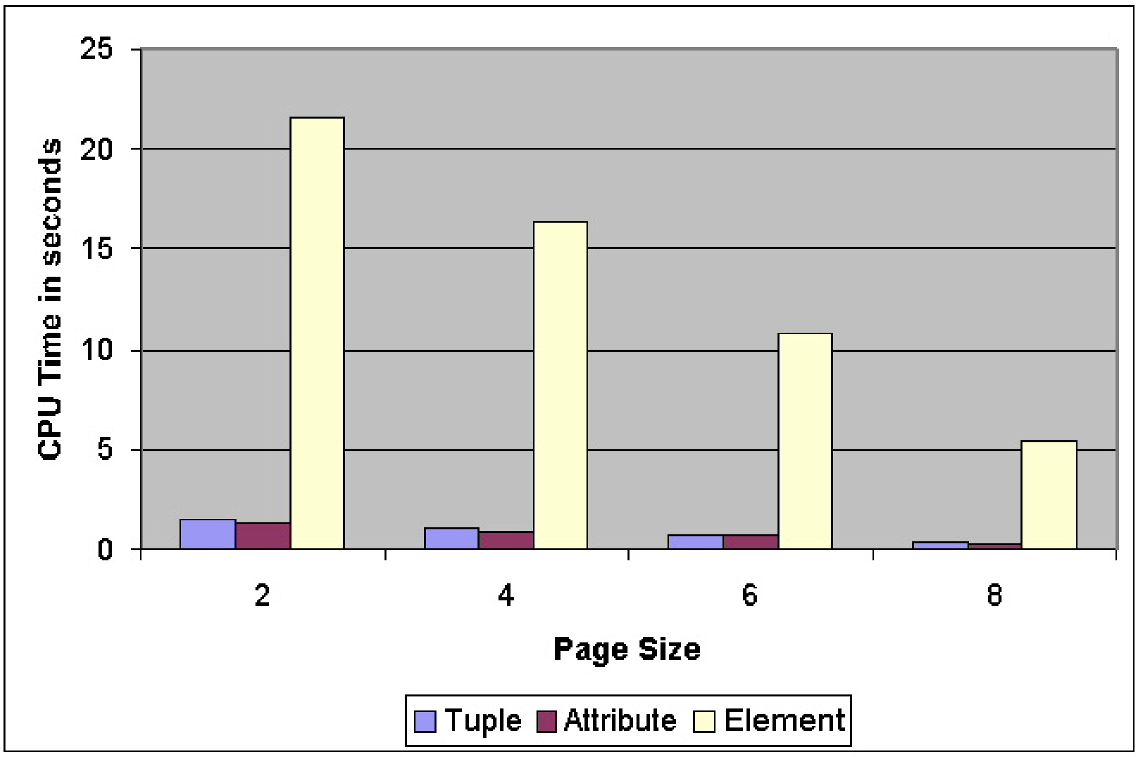

Figure 7.7: Vary the page size to 2, 4, 6, and 8. Fix # of tuples in each node at 105; fix # of attributes at 48; fix # of security levels at 16; fix user's clearance level. (Join Query).

Operational conditions for figure 7.8

Fragmentation Scheme: Tuple, Attribute, and Element

Relations: R_1, R_2, R_3, R_4, R_5, R_6, R_7, and R_8

Original Query: Join $R_1 \ldots R_8$

Criteria: Vary number of attributes to 8, 16, 24 , 32, 40, and 48; fix # of tuples at 1680; fix # of security levels at 16; fix user's clearance level; fix page size at 8.

X axis: # of attributes in all relations.

Y axis: CPU time in seconds.

7.6.4 Impact of varying the number of security levels

See section 7.5.4 for an explanation of this experiment and the results.

Operational conditions for figure 7.9

Fragmentation Scheme: Tuple, Attribute, and Element

Relations: R_1, R_2, R_3, R_4, R_5, R_6, R_7, and R_8

Original Query: Join $R_1 \ldots R_8$

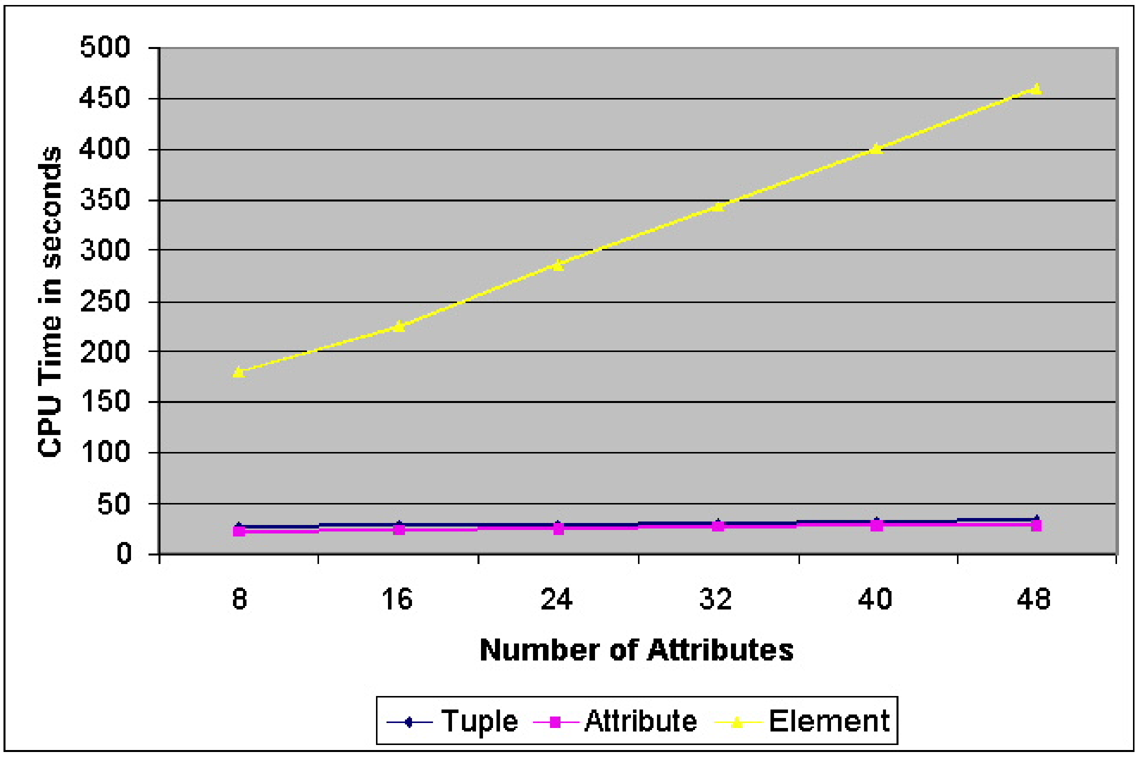

Figure 7.8: Vary number of attributes to 8, 16, 24 , 32, 40, and 48; fix # of tuples at 1680; fix # of security levels at 16; fix user's clearance level; fix page size at 8. (Join Query).

Criteria: Vary # of security levels to 1, 2, 4, 6, 8, 10, 12, 14, and 16. Fix # of tuples at 1680; fix # of attributes at 48; fix user's clearance level; fix page size at 8.
X axis: # of security levels in all relations.
Y axis: CPU time in seconds.

7.6.5 Impact of changing the number of users on CPU Load

See section 7.5.5 for an explanation of this experiment and the results.

Operational conditions for figure 7.10

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_1, R_2, R_3, R_4, R_5, R_6, R_7, and R_8
Original Query: Join $R_1 \ldots R_8$
Criteria: Vary # of concurrent users. Fix # of tuples at 1680; fix # of attributes at 48; fix user's clearance level; fix # of security levels at 16; fix page size at 8.
X axis: # of concurrent users.
Y axis: CPU time in seconds.

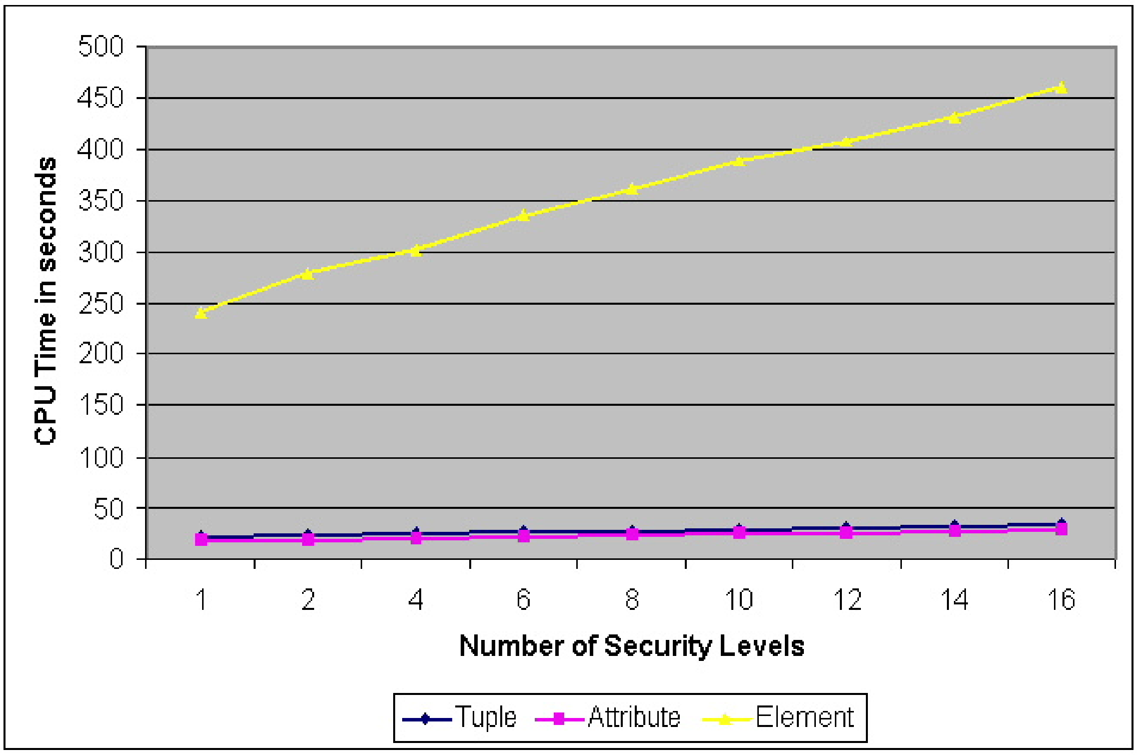

Figure 7.9: Vary # of security levels to 1, 2, 4, 6, 8, 10, 12, 14, and 16. Fix # of tuples at 1680; fix # of attributes at 48; fix user's clearance level; fix page size at 8. (Join Query).

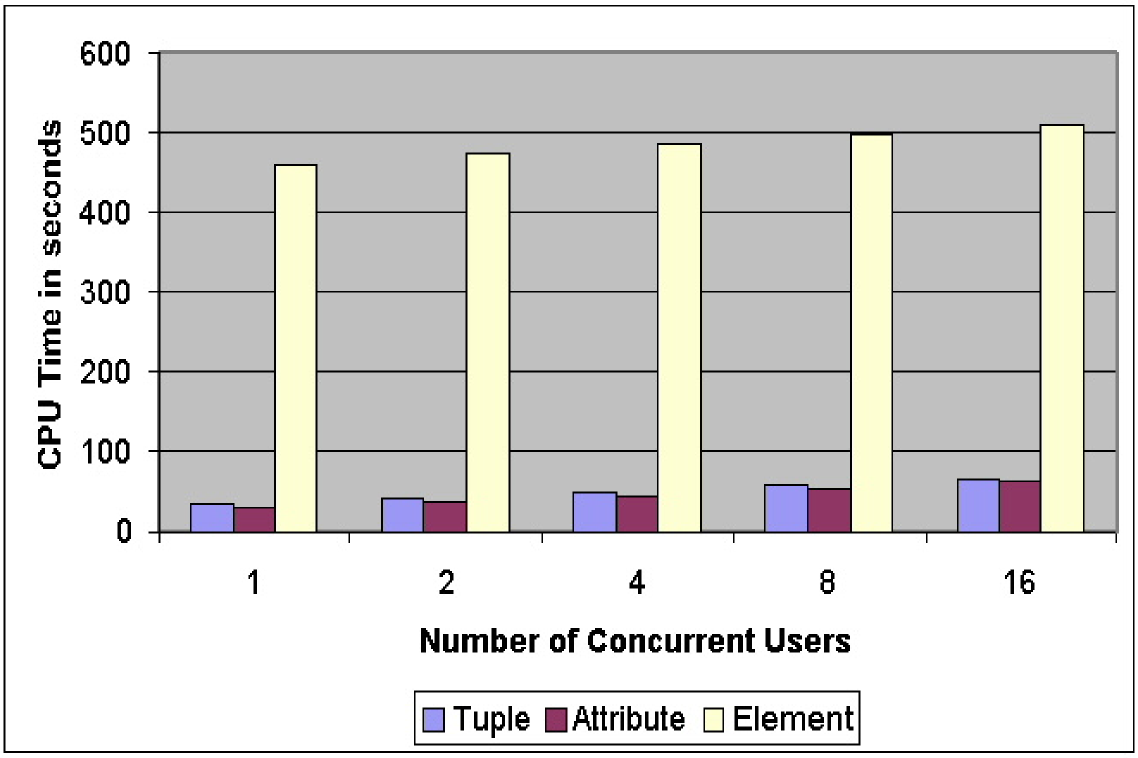

Figure 7.10: Vary # of concurrent users. Fix # of tuples at 1680; fix # of attributes at 48; fix user's clearance level; fix # of security levels at 16; fix page size at 8. (Join Query).

7.7 Updates

The experiments described in this section investigated the performance impact of an update operation on each of the three fragmentation schemes. The operation involved the *Encounter* relation. The Encounter relation has 512 tuples, 9 attributes, and 4608 elements. In the experiment, we executed UPDATE statements on all the date fields in the relation to 23/11/03 as the number of tuples was varied.

7.7.1 Impact of updating the encounter relation

Operational conditions for figure 7.11

Fragmentation Scheme: Tuple, Attribute, and Element
Relations: R_7
Original Query: UPDATE R_7; SET Date TO 23/11/03
Criteria: Vary # of tuples; fix # of concurrent users; fix # of attributes at 9; fix user's clearance level; fix # of security levels at 16; fix page size at 8.
X axis: # tuples updated.
Y axis: CPU time in seconds.

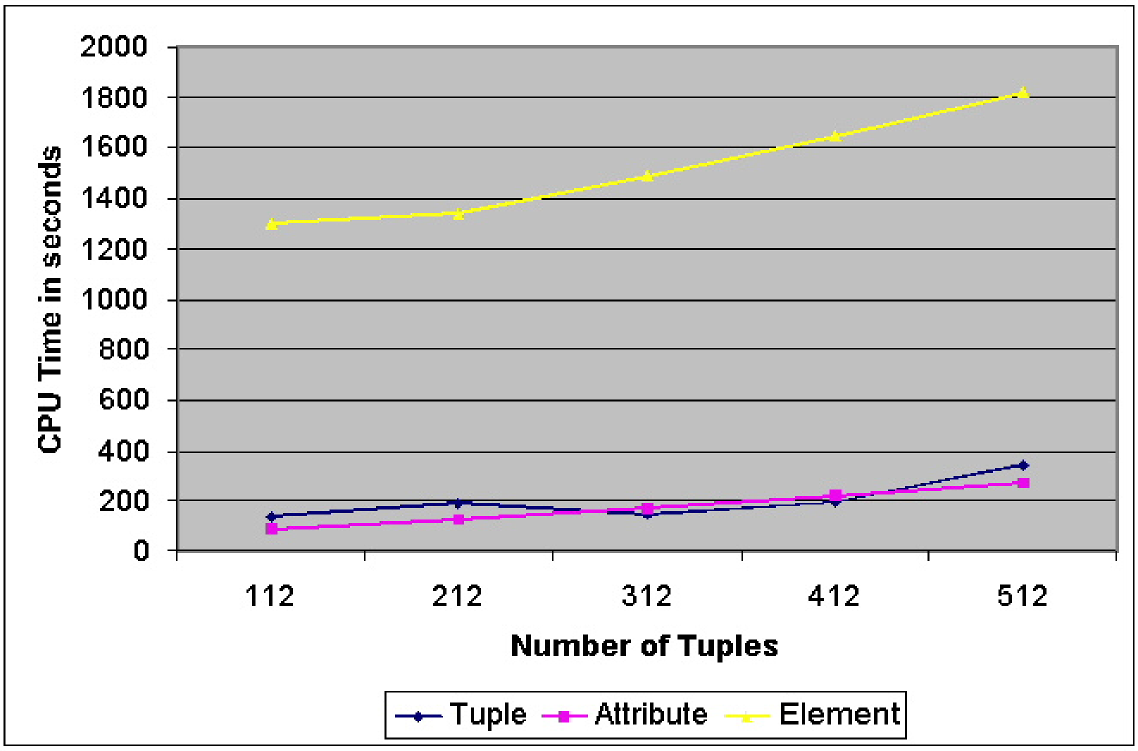

Figure 7.11: Vary # of tuples; fix # of concurrent users; fix # of attributes at 9; fix user's clearance level; fix # of security levels at 16; fix page size at 8.

Observation and explanation of results

Our observation reveals a clear disparity in CPU times between the fragmentation schemes for update operations. The CPU time for executing updates in the element-level fragmentation scheme was greater than for the attribute and tuple-level fragmentation schemes (see figure 7.11). This result shows that the CPU time consumed during update operations is significantly higher than that consumed during the query processing operations illustrated in our earlier experiments.

7.8 Analysis of experimental results

This section presents an analysis of how changes in the variables presented in our study affect the amount of operations performed by the prototype and the network traffic generated. Overall, the results have shown that the workload generated by each fragmentation scheme impacts the performance of the prototype.

7.8.1 Impact of varying the inhibitors on network performance

All three fragmentation schemes exhibited linear growth as the number of tuples were increased for both types of queries (see figures 7.1 and 7.6). However, the results also show that the attribute-level fragmentation scheme performs better than the tuple and element-level schemes. The worst performance was demonstrated by the element-level fragmentation scheme, which exhibited considerable performance degradation. The impact of network traffic on the performance degradation also appears to be significant; however, this is true for distributed database systems in general. The response time of the element-level fragmentation scheme is significantly higher than for the tuple and attribute-level schemes.

The page size is shown to have a significant impact on performance (see figures 7.2 and 7.7). This observation was very evident during the experiments when a number of "segmentation violation faults" (when a process overflows its stack) occurred as we decreased the page size below 8. The kernel recognised the violation but could not extend the stack size up to a configurable limit as is common in many operating systems. The Linux kernel does not keep track of each user thread's stack, so it was unable to perform this function. The implication of this observation for security is that the security domain of the MST prototype could become compromised by a Trojan Horse if page faults and stack overflows are a frequent occurrence.

There was evidence of performance degradation in all three schemes as the number of attributes was increased (see figures 7.3 and 7.8). The attribute-level scheme continues to exhibit superior performance relative to the tuple and element-level schemes. The worst case scenario was demonstrated by the element-level scheme. The comparatively modest impact shown as the number of attributes was increased left us wondering what would occur if the number of security levels was varied? Could the impact be more pronounced than, say, the impact of increasing the numbers of tuples or attributes?

Increasing the number of security levels gave a noticeable growth in response times. This is illustrated in figures 7.4 and 7.9 for both types of queries. By varying the number of security levels, we also vary the amount of 'reconstruction' required in computing a response. If one security level were to be used, the amount of reconstruction (vertical or horizontal) would be a fraction of that required for 16 security levels. The results show that the performance of the attribute-level scheme is superior to that of the tuple and element-level schemes; the element-level scheme continues to exhibit significant degradation. The results also show a more pronounced difference if a case with one security level is compared with one in which several security levels exist. Overall, it was found that a change in the number of security levels had a noticeable impact on the performance of all three fragmentation schemes.

As the number of users in the system increased, so did the overall CPU Load. The variation in CPU Load between higher level nodes and lower level nodes was clearly significant (see figures 7.5 and 7.10). The result reveals that nodes maintaining higher level fragments have an increased workload in most instances because they have the clearance to process fragments from several more nodes in the network. Contrast this with a Level-1 user who may want to reference all the distributed fragments but is unable to do so; consequently the workload of the Level-1 user's node is diminished because the node performs only a small fraction of the overall data processing task.

The cost of performing Join queries imposes additional overheads as was shown in the results from all previous 'Join' experiments, resulting in further performance degradation. We attribute the variations in response times between the three schemes to the overhead cost of executing the recovery algorithm and the network overhead. The element-level fragmentation scheme creates more fragments which must be combined to provide a response; this operation of combining fragments, especially for the element-level scheme, increases the network traffic significantly.

7.8.2 Impact of varying the inhibitors on front-end processing

As we increased the number of attributes in all three fragmentation schemes, the response times increased as a function of the number of attributes (see figures 7.3 and 7.8). Varying the number of attributes is implicitly the same as varying the size of the tuples; consequently, as the number of attribute fragments increase, so does the size of the records being retrieved in each fragment. There is a performance penalty inherent in processing large records. Furthermore, additional processing and network cost is also incurred through connection requests that must now be sent to more disjoint vertical fragments distributed across the network nodes. The results show that the response times of all three fragmentation schemes exhibited linear growth, increasing as the number of attributes were increased. The security label of each tuple, attribute, or element is also treated as an attribute in its own right, and therefore has an impact as we increase the number of attributes. The element-level scheme, however, continues to exhibit poor performance relative to the tuple and attribute-level schemes.

The results of varying the number of security levels also shows linear growth (see figures 7.4 and 7.9). This increase attests to the noticeable performance degradation in all three schemes, especially in the element-level scheme. Overall, the attribute-level scheme continues to exhibit superior performance relative to the tuple and element-level schemes.

The overall CPU Load increased significantly as the number of concurrent users in the system increased (see figures 7.5 and 7.10). However, the fraction of the overall load borne by higher level nodes was significantly higher than for lower level nodes. Again, this observation shows that the inherent advantages (i.e. access to more information) that come with the highest clearance in a security lattice also has its associated performance penalties. Essentially, what this indicates is that higher level nodes may often examine more data during query processing than lower level nodes, which are restricted by security policies from doing so.

7.8.3 Impact of executing an UPDATE operation

The need to coordinate distributed transactions throughout the network imposes a significant cost on the performance of all three fragmentation schemes. Although the prototype uses a variant of the two-phase commit protocol that allows local machines to fail without forcing a 'global rollback', significant performance gains are not evident. The relative performance of the three fragmentation schemes is unchanged in this experiment — with the element-level fragmentation scheme still exhibiting significant performance degradation. The Stargres Visual DBA monitoring tool also revealed a significant demand in network resources as the number of local DBMSs to which data must be distributed increases. This observation was particularly evident for the element-level fragmentation scheme.

The amount of operations performed by the prototype is largely determined by the fragmentation scheme. The more fragments and nodes that must be contacted in response to a query, the higher the processing cost. This cost rises significantly for the element-level fragmentation scheme and is partly responsible for the high performance degradation that besets this approach. The implication for this study is that the fragmentation scheme that processes the most amount of fragments is burdened as the various disjoint fragments are assembled in response to a user's query.

7.9 Summary

Through a number of experiments, this chapter compared the performance of all three fragmentation schemes of the MST prototype. We investigated the performances by varying the numbers of tuples, attributes, security levels, user's clearance level, and page size using the Select-All and Join queries. All three fragmentation schemes exhibited linear growth as the numbers of each variable were increased. However, the attribute-level scheme performed better than the tuple and element-level schemes. The element level scheme exhibited the worst case of performance degradation due to the numerous Union and Join operations required to reconstruct its response relations. The attribute-level scheme exhibited the best performance for both types of queries. The impact of varying the number of attributes and the number of security levels was not as significant as the impact of varying the number of tuples. The overhead imposed by network traffic continues to have a significant impact on the response times; this impact increases with the number of nodes that must be contacted in response to a user's query. The processing cost is directly affected by the fragmentation scheme; the scheme with the greatest number of fragments to process imposes a higher load on the system. In addition, the type of query processing — be it a Join or a Union — makes the impact on the front-end more pronounced.

The processing cost imposed by Join query operations had a significant impact on system performance, resulting in an increased response time for all three schemes. The increased response times compared unfavourably to the Select-All query operations. The overall CPU Load also increased significantly as the number of concurrent users increased.

The cost of performing updates, although significantly higher than that for query processing, did not contradict the growth pattern observed for all three fragmentation schemes. The observed increase in processing activity during updates, particularly for the element-level fragmentation scheme, amplified the performance degradation of the element-level scheme to a level that may be unacceptably high for most database processing environments.

In chapter 8, we present some conclusions from the adaptation effort of the Stargres DBMS and the performance investigations presented in this chapter. We also suggest a few areas that are in need of further work to enhance the MST prototype.

Chapter 8

Conclusions and Further Research

The development cost and complexity of MLS/DBMS software is cited by most vendors as a major factor in their decision to cease supporting these software systems. Some researchers also argue that the era of MLS/DBMS research and development has come to an end, but only a few contend that the security concerns that instigated the early research efforts have ceased to exist. In the presence of these concerns, it is important that we continue to consider less complex and more cost effective alternatives to implementing multilevel security in DBMSs. This thesis proposed one alternative approach that augments an existing COTS DDBMS product with multilevel security, and that has been prototyped in a limited adaptation effort.

In adapting the COTS DDBMS to provide multilevel security at the tuple, attribute, and element-level fragmentation schemes, we encountered several research issues that had a direct or indirect impact on the performance of each fragmentation scheme. In our approach, we targeted and adapted specific modules in the COTS DDBMS, specifically those modules in the query processing pipeline that are responsible for transforming and/or routing an SQL query. The following section summarises our contributions, draws some conclusions from our work, and discusses the effort involved and issues raised in adapting a COTS DBMS to MLS/DBMS. Section 8.2 presents a brief MLS analysis of our prototype. Section 8.3 points to areas requiring further research.

8.1 Contributions and conclusions

The major contributions of our work are related to the adaptation of the Stargres system, and to the performance experiments that were instrumented using the system. The performance experiments revealed information about the impact of different variables on the performance of MLS/DBMSs implemented under three fragmentation schemes. It adds to the work of Thuraisingham and Kamon [77], described in chapter 3, in the sense that the impact of more variables was placed under observation, and at different levels of granularity.

8.1.1 Summary of contributions

Collectively, the contributions of this thesis demonstrate the feasibility of adapting a COTS DDBMS product to provide multilevel security. We also illuminated performance arguments that must be considered by organisations that would wish to embark on a similar adaptation effort. A summary of the main contributions of this thesis is as follows:

- Our proposed approach contends that the query processing techniques developed for transforming global queries into query fragments in distributed databases can also be used to efficiently implement multilevel security. To this end, we augmented a COTS DDBMS code with multilevel logic such that the distribution of relation fragments and sub-queries to relevant nodes is determined on the basis of security level only. This ensures that sub-queries en-route to unauthorised nodes are intercepted and dropped by the Distributed server.
- We developed a prototype that utilises the query decomposition and recovery feature of distributed databases to recover relation fragments distributed across several nodes in a network. We also show that the same techniques used for adapting the Stargres COTS product, in which critical modules were augmented with MLS code, can also be utilised by subscribers to MLS policy who no longer enjoy direct vendor support.
- We investigated the performance of the three fragmentation schemes supported by the MST prototype to determine the impact of varying properties such as the number of tuples, number of attributes, number of security levels, number of nodes, user clearance, and page size on the front-end and the network for a Select-All and Join query. A full analysis of experimental data was given in chapter 7; however, a summary of the main findings from the performance study is as follows:
  - All three fragmentation schemes exhibited linear growth as the number of tuples and attributes were increased for the Select-All and Join queries; however, the rate of performance degradation was more evident in the element-level fragmentation scheme. The performance of the attribute-level fragmentation scheme was comparatively superior to the tuple and element-level schemes (see sections 7.5.1 and 7.5.3).
  - Performance is affected by the page size in all three fragmentation schemes; it deteriorates as the page size decreases due to increased paging activity (see section 7.5.2).
  - A variation in the number of security levels affects the performance of all three fragmentation schemes. As the number of security levels increases, so does the amount of reconstruction needed to generate a response set, which in turn increases performance degradation. The degradation in performance is linear with

respect to the number of security levels. Its effect is more evident in the element-level fragmentation scheme (see section 7.5.4).

- The number of concurrent users in the system also directly impacts performance. As the number of users increases, so does the performance degradation. The degradation is more pronounced at the element-level of granularity if all the concurrent users are logged-in at System-high, because there are more fragments being processed. This contrasts with a scenario where all the concurrent users are logged-in at System-low and can only access a fraction of the overall number of fragments available (see section 7.5.5).

8.1.2 Conclusions

Our approach provides organisations with a strategy for adapting their existing COTS DBMSs to provide multilevel security. An MLS/DBMS can be deployed without the prohibitive cost of investing in custom designed MLS/DBMSs. The main finding described in the previous section will also provide these organisations with sufficient information to determine if a particular type of system fits their requirements. There are several possible system types that can be created as a result of the information provided by the study. For example, an organisation could choose to implement the element-level fragmentation scheme but with limits placed upon the number of security levels. We draw the following conclusions about the effort and the performance experiments that were conducted.

- The performance variations between the three fragmentation schemes translates into large cost differences. Based on the results of the performance experiments, the attribute-level fragmentation scheme provides a better implementation option, that is, if the main indicator of a system's suitability is its response time.

- There is a significant network overhead, increasing as the number of fragments being processed increases. This is particularly true for the element-level fragmentation scheme. An organisation's network architecture and configuration should be a factor in deciding which scheme to implement.

- Adapting an existing COTS DBMS, although a non-trivial task, can be accomplished with modest resources and within a reasonable time. However the complexity of the task and the possibility of success or failure is for the most part determined by the internal structure of the target software. Overall, the AUF facility and the Distributed server modules that were adapted, consisted of more than 57 functions with approximately 4,860 lines of code in total. It should be recognised that not all COTS DBMS products will be suitable candidates for the kind of adaptation effort proposed in this thesis.

- The COTS DBMS products individually do not provide high assurance solutions, and the successful integration and operation of the constituent components was highly challenging. We encountered the following issues during the adaptation and configuration effort:
  - The identification of a suitable and stable open source DDBMS was not as effortless as we had anticipated. Many of the open source DBMS products that are currently available do not possess a distributed feature set; distribution was a necessary requirement in our design. Although Mnesia and Mariposa (see section 5.2) were highly documented systems, Mnesia did not support fragmentation — a key requirement in our target system; for Mariposa, the issue of objects migrating to other nodes presented a security dilemma; controlling the movement of objects was necessary to ensure security. After identifying Stargres as a suitable target for adaptation, the next challenge was to successfully disentangle the code to identify the modules that could be modified. Due to the size of the DDBMS, it was impractical to undertake this task manually, and hence a SWAG reverse engineering toolkit (see section 5.2) was utilised to identify the modules and their interdependencies.
  - The configuration of the network was accomplished using the LinuxConf utility; this was for the most part, rather straightforward. However, by default, the resolution of Internet Protocol (IP) addresses and host names was managed by a Domain Name Server (DNS) outside of our experimental cluster. Dropping this role and assigning it to the Distributed server node required a re-configuration of the entire subnet such that 1) the Distributed server is not also acting as the Domain Name Server for any node outside the cluster, and 2) nodes in the cluster cannot contact a DNS outside the cluster to resolve host names. Our strategy was to separate the cluster traffic from the pre-existing network traffic by routing them through separate switches, thus providing our cluster with its own dedicated switch. This part of the task was easily accomplished. A problem emerged because of a requirement in the Linux DNS configuration that the hostnames and IP addresses be listed in the host files of a primary and secondary DNS. Although the traffic was separated, a node in the cluster that is unable to contact the Distributed server node within a specific time frame will seek resolution at a secondary DNS outside of the cluster, and, if that fails, it will default to other nodes further up the network tree. This situation was an obvious security flaw that required the editing of a file located at the /etc/host directory of each node in the cluster.
  - A significant part of the Stargres DBMS source code is uncommented, and, in some cases, references to external files were not explicit. This introduced signifi-

cant compilation problems during the adaptation effort, resulting in delays. The most significant of the problems relates to the cascading nature of logic errors to other modules after a function has been altered. For example, when the Register node command was modified to require a security label entry, it generated several referential errors and memory consistency errors in the RAM and CDM modules of the Distributed server. This remains one of the major challenges involved in the adaptation of a COTS DBMS software, or any other large software system for that matter.

8.2 Security analysis of MST

The design of our proposed model (see chapter 4) was influenced largely by our thesis – the adaptation of a COTS distributed DBMS for MLS. This resulted in a tradeoff between security (covert channels) and the practicality of implementation. The complexity and size of modern DBMSs also means that it is impractical for an entire DBMS system to be subject to an exhaustive security analysis. Consequently, we only engaged in a limited MLS analysis of the prototype's trusted component, i.e., the Distributed server – recognising that covert channels can never be totally eliminated in many practical high assurance systems. All the other components of the prototype are untrusted (i.e., they are not exempt from mandatory access control). The Distributed server mediates all accesses made by subjects to objects using a set of mechanisms and modules that control access to objects, assign security levels, maintain subject and object attributes, and perform logging functions. Compliance with MLS policy, ensuring that information flows within the MST system does not lead to a violation of the BLP's two security properties, is considered in the context of four scenarios. These scenarios include the Insert, Retrieve, Update, and Delete operations. For each of these scenarios, we identify possible threats, their sources, and their impact.

8.2.1 Processing Scenarios

The following scenarios are typical of the kinds of operations that are requested of the MST prototype by users. An analysis of the information flows during these operations is necessary to determine the degree to which the prototype complies with MLS policy, particularly the BLP properties.

- **Insert operations.** Users may only insert tuples if they are logged-in at System-low. System-high users cannot insert tuples unless their security level is downgraded to System-low.

- **Retrieve operations.** Users can only retrieve data classified at or below their security levels. A user logged-in at System-low cannot view data classified at System-high.

- **Update operations.** Users may only update tuples at their security levels only. Upwards information flow is permitted.
- **Delete operations.** Users may only delete tuples if they are logged-in at System-low. System-high users must downgrade their security levels to System-low in order to delete a tuple.

8.2.2 Threats and their sources

Threats to the security of the MST prototype can be in different forms. We will only consider threats arising from a denial of service attack, the insertion of a Trojan Horse in a data communication device, and the presence of covert channels in the Distributed server. These threats can result in sensitive data being made unavailable to authorised users, the disclosure of sensitive information, or the illegal modification of data. The threats can also occur intentionally or accidentally, but whatever the cause they are all damaging if they are realised.

1. **Denial of service attack.** The database could be monopolised in such a way that legitimate users cannot access complete records. For example, during insertion operations, a user logged-in at Level-1 could launch this type of attack by inserting a place holder field in a record and assigning it a Level-16 classification, thereby preventing all users (except Level-16 users) from getting complete answers to their queries (see 3.4.4).
2. **Communication attack.** The Distributed server has the special privilege of handling communication between all the nodes in the cluster. This privilege makes it an ideal target for attack. The communication link could be tampered with by inserting a Trojan Horse at the switch designed to intercept and perhaps modify data communications between the nodes in the cluster. This threat is very much implementation-dependent and any further analysis would be highly speculative. However, it is clear that, wherever possible, the LAN devices and communication links should be designed to minimise the threats.
3. **Trojan Horse attack.** Despite the mandatory access control mechanism of the Distributed server, it is still possible for a Trojan Horse running at the Distributed server to covertly leak information. The MST prototype, like all multilevel systems, requires information flow from System-low to System-high. Furthermore, reliability and atomicity[1] of transactions are also integral components of high-assurance computing. In a COTS DBMS, a user/process would wish to receive an acknowledgement of a successful update. Without acknowledgements, necessary data may be written over, or may

[1]All or nothing without the appearance of interruptions.

be lost during a crash. In a non-secure DBMS like the Stargres, there is no problem with acknowledgements; however, in a secure DBMS like MST, acknowledgements can allow a covert channel to exist between System-low and System-high. In the following, we show how a Trojan Horse can exploit the Distributed server and cause the release of sensitive information.

(a) A Trojan Horse running at the Distributed server could continuously monitor inbound/outbound traffic at the NSV module of the Distributed server, noting the type and classification label of each request, and transmitting this information to a second Trojan Horse at a System-low node which could then store the data for later use. Recall that the NSV module is aware of the address of all the nodes in the cluster and will only contact them if authorised to do so by the Distributed server.

(b) The contents of the CDB, specifically the classification of data objects and the clearance of users, which unlike user passwords are unencrypted, could be attacked by two cooperating Trojan Horses. A legal query could be launched by the Trojan Horse running at the Distributed server against a specific CDB table to retrieve its entries; this could then be written to a System-low node using the same technique described in (a) above. This vulnerability, as we stated earlier, is a consequence of the tradeoffs that had to be made in building the MST prototype from several COTS components. Furthermore, our design assumed that, because the Distributed server is trusted, additional security measures were perhaps unnecessary to protect it from compromise. The basis of this assumption is that the Distributed server will not fall victim to a Trojan Horse attack; however, this is highly speculative. Further security measures could be employed to protect the contents of the CDB from such an attack, but the performance penalty of applying these measures must also be considered.

(c) A user at System-low can infer high-level information (Allen and Reuter's salary) by formulating a query such as: **If (Allen's salary > Reuter's salary) return 1, else return 0**. This query will modulate its output based on the value of System-high data.

8.2.3 Impact of security threats

Next, we consider the impact of the threats identified in the previous section. If threat #1 is successful, all (except Level-16 users) will be denied access to information to which they are authorised, leading to a significant degradation in quality of service. Undoing the impact of this damage will depend to a large extent on the number of data items that are affected and that therefore must be re-classified. Should threat #2 become successful, adherence to the BLP's security policies will be gravely compromised as the Trojan Horse could modify the security labels of fragments, the clearance of a user making a request, or even the destination of a query or query response.

8.3 Further research and future issues

The MST prototype could be further enhanced to improve its efficiency and performance. This section suggests two ways in which this might be accomplished, and points to two issues which, although peculiar to distributed databases in general, have a particular implication on our architecture and require further research. These issues include 1) the necessity for novel query optimisation techniques to improve the performance of the prototype, and 2) the maintenance of security label information. Further work on enhancements to the prototype could therefore proceed in these directions.

- A useful enhancement to the MST prototype would be the development of query optimisation techniques that consider the additional complexity inherent in MLS/DBMS systems. This is of particular importance as the performance of the fragmentation schemes, especially the element-level scheme, is in need of substantial improvement. Innovative approaches to query optimisation are essential if the prototype is to achieve acceptable performance and efficiency. The optimisation must take into account the extra communication costs of moving data from site to site, as this is considerably more complex in a non-replicated distributed DBMS network. Replicated and centralised DBMSs can exploit the advantages of locality of reference to improve the speed of processing a request. The strength offered by relational expressions is at a sufficiently high level to make query optimisation feasible. This contrasts with non-relational systems where user requests are low level and optimisation is done manually by the user. Semantic query optimisation techniques, that specify constraints on the database schema, used in combination with other optimisation techniques, could be exploited.

- In the current architecture, security classifications are stored alongside the data items, thus requiring additional effort to suppress them in the response relation. This solution is sub-optimal and further work is required to 1) determine how labels could be stored separately from data, and 2) decide how security labels can be associated with their corresponding data without disclosure. The separate storage of classification information could introduce an additional performance overhead. In addition, the table maintaining the security label information could become the target of attacks.

Glossary

The purpose of this glossary is to explicitly define some of the terms that occur most frequently in this thesis. It is not a comprehensive list of information security terminology. The definitions were taken mainly from *Information security: an integrated collection of essays* [2], *Computer security basics* [72], *Computer security* [38], *An introduction to database systems* [24], and *Stargres release 3.2* [55, 56].

***-property**	A Bell-LaPadula security model rule allowing a subject write access to an object only if the security level of the subject is dominated by the security level of the object. Also known as the *no write down* (NWD) rule.
A1	The Highest level of trust described in the Orange Book.
API	Application program interface. System access point or library function that has a well-defined syntax and is accessible from application programs or user code to provide well-defined functionality.
Arity	The number of attributes in a relation(s).
Assurance	A measure of confidence that a system's security features have been implemented and work properly. Assurance is one of the primary issues addressed by the Orange Book.
Attribute	Each column in the table contains the values for a specific attribute of the relation. A column name is known as an attribute.
Audit	To record independently and later examine system activity (e.g., logins and logouts, file accesses, security violations).
AUM	Administrative utilities module. A module of the AUF that, in addition to performing syntax checks, provides a number of utilities for carrying out administrative tasks such as registering nodes, creating user accounts, creating or dropping databases and tables, schema definition, shutting down the server, and so forth.

Authentication	The process of proving that a subject (e.g., a user or a system) is what it claims to be. It is a measure used to verify the eligibility of a subject and the ability of that subject to access certain information.
Base Relation	A named relation in permanent storage that is not a derived relation (i.e., base relations are autonomous). In practice, base relations are those relations that have been judged to be sufficiently important (for the database at hand) that the designer has decided it is worth giving them a name and making them a direct part of the database – as opposed to other relations that are more ephemeral in nature, such as the result of a query [24].
BLP	Bell-LaPadula model. The computer security policy model on which the Orange Book requirements are based. It is a formal description of the allowable paths of information flow in a secure system. The goal of the model is to identify allowable communication where it is important to maintain secrecy.
Cardinality	The number of tuples in a relation.
CCM	Concurrency control manager. A module of the TMF; it is responsible for making sure that transactions are executed separately and independently. It does so by acquiring locks on appropriate pieces of data from the locking table that is stored in memory.
CDB	Coordinator database. The database that maintains the catalogs that the Distributed server uses to keep track of objects within the cluster.
CDM	Coordinator database module. A Distributed server module that is responsible for managing the CDB which maintains the catalogs that the Distributed server uses to keep track of all the databases in the cluster.
Classification	The hierarchical portion of a sensitivity label (the non-hierarchical portion is called the "category set" or the "compartments"). A classification is a single level in a stratified set of levels. For example, in a military environment, each of the levels UNCLASSIFIED, CONFIDENTIAL, SECRET, and TOP SECRET is more trusted than the level beneath it.
Clearance	A representation of the sensitivity level (the classification and the categories) associated with a user in a system supporting mandatory access controls. A user with a particular clearance can typically access only information with a sensitivity label equal to or lower than the user's clearance.

Confidentiality	The assurance that information is not disclosed to unauthorised entities or processes.
Covert Channel	A communications channel not normally meant for information flow but that could nevertheless be used by a process to signal information in a way that violates a system's security policy.
COTS	Commercial off-the-shelf product. A combination of hardware, firmware, and software that can be purchased commercially.
CQIM	Client/Query interface module. A module of the AUF; it provides the interface that facilitates the interaction between users and the DBMS. It also imposes restrictions on the types of commands that can be executed by non-administrative users.
DAC	Discretionary Access Control. A means of restricting access to objects based on the identity of users and/or groups to which they belong. DAC is the most common type of access control mechanism found in trusted systems.
Database	An organised collection of logically related records or files.
DBA	Database Administrator. The technical person responsible for implementing and administering a database management system.
DBMS	Database Management System. The collection of hardware and software that organises, manages, and provides access to a database.
DDBS	Distributed Database. A collection of multiple, logically interrelated databases distributed over a computer network.
DDBMS	Distributed Database Management System. The software system that permits the management of the DDBS and makes the distribution transparent to the users.
DDL	Data Definition Language. A language used by database administrators to create, store, and manage data in a database environment.
DML	Data Manipulation Language. A language that allows users to interrogate and access a computerised database system using English-like statements.
DMP	Data Manipulation Language Precompiler. A Distributed server module that is responsible for compiling global DML statements.

Domain	A set of data values from which specific attributes of specific relations draw their actual values, e.g., the domain of the age attribute might be integers from 0 to 120.
DOPM	Distributed query optimiser module. A Distributed server module that is responsible for generating global query execution plans.
DQEM	Distributed query execution engine. A Distributed server module that provides the functions used for creating and updating relation fragments, deleting databases, and transmitting sub-queries to local sites for processing.
DQM	Distributed query handler. A Distributed server module that is responsible for performing syntax and verification checks on queries.
Element	An indivisible item of data.
Ethernet	A local area computer network designed on the principle that one computer wishing to communicate with another, broadcasts onto the network. Acknowledgement establishes the link.
Firewall	The generic name for any security system protecting the boundary of an internal computer network. A *bastion host* is a computer system with strong security as it is exposed to the outside world.
GDC	Global DDL compiler. A Distributed server module that is responsible for compiling global DDL statements.
Instance	A set of tuples in the database for a specific relation at a specific time.
Integrity	A security principle that keeps information from being modified or otherwise corrupted either maliciously or accidentally. Integrity protects against forgery or tampering. Synonymous with *accuracy.*
Kernelized	Kernelized Architecture. A multilevel database is partitioned into single-level databases, which are then stored separately. In this architecture, there is a separate DBMS for each security class.
LMM	Log manager module. A module of the TMF that is responsible for logging every operation executed in the database. It does so by storing the log on disk through the buffer manager. The operations in the log are stored as DBMS commands. Thus, in the case of a system crash, executing every command in the log will bring back the database to its last stable state.

Login	The process of identifying oneself to, and having one's identity authenticated by, a computer system.
MAC	Mandatory Access Control. Unlike discretionary access control (DAC), which allows users to specify, at their own discretion, who can and cannot share their files, mandatory access control puts all such access decisions under the control of the system.
Metadata	A special database, also referred to as a data dictionary, containing descriptions of the elements, (e.g., relations, domains, entities, or relationships) of a database.
MST	Multilevel Stargres prototype. Our proposed multilevel secure database management system.
Multilevel	Used to describe data or devices. Multilevel security allows users at different sensitivity levels to access a system concurrently. The system permits each user to access only the data that he or she is authorized to access. A multilevel device is one on which a number of different levels of data can be processed.
mSQL	Mini SQL. Open Source Database distribution.
MySQL	Open Source Database distribution.
NSV	Network server module. A module of the TMF facility that adds the network communication element to Stargres. Queries destined for remote nodes are routed to this module by the QRM module. It is also responsible for opening and closing network connections.
Object	From the Orange Book definition: "A passive entity that contains or receives information. Access to an object potentially implies access to information it contains. Examples of objects are records, fields, files, and programs."
OPM	Query optimiser module. A module of the QPF; it is responsible for optimizing queries. It does this by analyzing the processed query to see if it can take advantage of any optimizations that will allow it to process the query more quickly. It uses indexes whenever possible and uses the most restrictive index in order to first eliminate as many rows as possible as soon as possible.
PC	Personal Computer. A moderately priced microcomputer system intended for personal use rather than commercial purposes.

Polyinstantiate	A technique used by the DBMS to prevent inference. It allows the DBMS to contain multiple instances of the same data item, each one having its own classification level. In other words, different tuples with the same key can exist at different classification levels if for example a high-classified row already exists and a low-classified user requests the insertion of a new row having the same key [15].
Primary-Key	A unique set of attribute(s) which identify an individual tuple in a relation.
QPF	Query processor facility. A facility that is responsible for parsing and optimizing DML statements. It consists of the DMP module, the DDL compiler (DDC), the query parser (QPM), the query preprocessor (PPM), the query optimiser (OPM), and the Query execution engine (QEM).
QPM	Query parser module. A module of the QPF; it is responsible for parsing queries.
Query	To ask for information. To make a request or interrogate a database for information.
QR	Query Response. The answer(s) returned to a user by the database in response to the user's query.
QRM	Query router module. A module of the AUF that is responsible for directing DDL and DML queries to the appropriate pipelines for processing.
QTM	Query transformer module. A Distributed server module that is responsible for decomposing global queries against relations into sub-queries against relation fragments.
RAM	Remote access module. A Distributed server module; it uses information stored about nodes by the CDM module during the registration of nodes, and the creation of databases, tables, and user accounts to distribute fragments to relevant nodes.
Reference Monitor	From the Orange Book definition: "An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects".
Relation	A set of values collected as information about something of interest to the user. Each relation is represented by a table with a name (the relation or table name).

Response Time The time it takes the computer system to react to a given input, e.g., a query. It is the interval between an event and the system's response to the event.

RPC Remote Procedure Call. A client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing an application to be distributed over multiple heterogeneous platforms. It insulates the application developer from the details of the various operating system and network interfaces.

Schema The definition of a table (i.e., its name and attribute names) is called the schema.

Secrecy A security principle that keeps information from being disclosed to anyone not authorized to access it. Synonymous with *confidentiality*.

Security Freedom from risk or danger. Safety and the assurance of safety.

Security Domain A collection of nodes adhering to a particular security policy. A security domain could have both trusted and untrusted nodes.

Security Kernel From the Orange Book definition: "The hardware, firmware, and software elements of a Trusted Computing Base that implement the reference monitor concept. It must mediate *all* accesses, be protected from modification, and be verifiable as correct.

Security Level A representation of the sensitivity of information, derived from a sensitivity label (consisting of classification and categories).

Security Model A precise statement of the security rules of a system.

Security Policy From the Orange Book definition: "The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information".

Simple security property A Bell-LaPadula security model rule allowing a subject read access to an object only if the security level of the subject dominates the security level of the object. Also known as the *no read up* (NRU) rule.

Snapshot A named derived relation. Like base relations, they are also relation variables. However, a snapshot is real, not virtual – i.e., it is represented not only by its definition in terms of other named relations, but also (at least conceptually) by its own separate data [24].

SSN	Social Security Number. A unique identifier used by the U.S. government to identify its citizens. It consists of nine digits, commonly written as three fields separated by hyphens: AAA-GG-SSSS. The first three-digit field is called the "area number". The central, two-digit field is called the "group number". The final, four-digit field is called the "serial number".
SQL	Structured Query Language. A database query language that was adopted as an industry standard in 1986.
StarQL	Stargres Query Language. The query language of the Stargres DBMS. It is an extension of the standard SQL.
Subject	From the Orange Book definition: "An active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state".
System high	The highest security level supported by a system at a particular time or in a particular environment.
System low	The lowest security level supported by a system at a particular time or in a particular environment.
TFE	Trusted Front-end. A front-end application that has been evaluated for its adherence to the security policy and whose incorrect or malicious execution is capable of violating system security policy.
Trap Door	A concealed entry-point to software in a computer system that allows normal system protection to be circumvented.
Trojan Horse	An independent program (software) that appears to perform a useful function while doing something else (usually destructive).
Trust	Reliance on the ability of a system to meet its specifications.
TCB	Trusted Computing Base. The totality of protection mechanisms within a computer system - including hardware, firmware and software - the combination of which is responsible for enforcing a security policy.
TCSEC	The Department of Defense Trusted Computer System Evaluation Criteria (or Orange Book). The document that describes the evaluation criteria used to assess the level of trust that can be placed on a particular computer system.
TDI	The Department of Defense Trusted Database Management System Interpretation (or Lavender Book). The document that interprets Orange Book requirements for database management system products.

TMF	Transaction management facility. A facility that is responsible for ensuring that a transaction is logged and executed atomically. It does so with the aid of the concurrency control manager (CCM), the transaction manager (TMM), and the log manager (LMM).
TMM	Transaction manager module. A module of the TMF; it is responsible for making sure that a transaction is logged and executed atomically. It does so through the aid of the log manager and the concurrency control manager.
TNIU	Trusted Network Interface Unit. A trusted mediation device placed between each system and its network connection for the purpose of permitting communication between machines belonging to the same security level.
Trusted System	A system designed and developed in accordance with Orange Book criteria and evaluated according to those criteria.
Tuple	Each row in the table contains the values for a member of the relation set. A row is known as a tuple.
Uniprocessor	A computer system with only one processor on its system board.
User	A person or a process who accesses a computer system.
UFE	Untrusted Front-end. A front-end application that has not been evaluated or examined for adherence to the security policy. It may include incorrect or malicious code that attempts to circumvent the security mechanisms.
View	A named derived relation that results from a relational query. Like Snapshots and Base relations, they are also relation variables. Views are also virtual – they are represented within the system solely by their definitions in terms of other named relations [24].
VPN	Virtual Private Network. A secure connection between gateways in two subnets that are not directly connected. All traffic between the subnets occurs through the gateways where cryptographic protection is added to extend the security perimeter.

Appendix A

Database Structure

The following relation schema represents the structure of two relations from the *Hospital* database. Figure A.1 shows a subset of the *Department* relation, Figure A.2 shows a subset of the *Nurse* relation, and a subset of the *Patient* relation is shown in Figure A.3.

- Department(DN, Dept_Name, Manpower, Dept_Location, Manager)
- Nurse(N_SSN, N_Name, N_DOB, N_DOR, N_Salary, N_Specialty, N_Pager, DN, Manager)
- Patient(P_SSN, P_Name, P_DOB, P_Status, P_Tel., INS_Name, PHY_Name, EC_Name, Patient #)

DN	**Dept_Name**	**Manpower**	**Dept_Location**	**Manager**
D10	Records	43	241 East Annex	Cantor
D11	Obstetrics	13	601 Main Building	Baker
D12	Laboratory	8	202 Drew Building	Ebert
D13	Geriatrics	12	B16 Basement	Xavier
D14	Paediatrics	23	512 West Wing	Gerald
D16	Ambulance	12	Emmergency Wing	Leach
D17	Cathering	9	Kitchen	Sachs
D18	Radiology	4	409 Drew Building	Kain
D19	Tertiary Care	11	121 South Wing	Usher
D21	Orthopaedics	9	714 South Wing	Zalkin
D22	Neurology	17	305 Outpatient Wing	Beaver
D23	ICU	24	513 West Wing	Hagger
D25	Pharmacology	10	216 East Annex	Carter
D26	Pathology	7	234 West Wing	Marshall
D28	Supplies	6	B23 Basement	Richards
D29	Management	4	120 Main Building	Patterson

Figure A.1: The Department relation.

N_SSN	N_Name	N_DOB	N_DOR	N_Salary	N_Specialty	N_Pager	DN	Manager
200000	Abbott	110466	170788	23,000	Surgical	224 165	D11	Cantor
200050	Allen	240371	260793	19,000	Gerontology	321 675	D13	Xavier
200100	Ball	090562	250785	20,000	Midwifery	641 876	D11	Cantor
200150	Barclay	210474	230797	17,000	Psychiatry	986 521	D22	Beaver
200200	Chapman	090171	120795	16,900	Paediatrics	571 443	D14	Gerald
200250	Cheadle	230575	250798	16,600	Surgical	876 453	D22	Beaver
200300	Darby	250961	160785	21,000	Perinatal	721 887	D11	Baker
200350	Dorman	040668	220791	20,000	Child	443 654	D23	Hagger
200400	Evans	231269	280795	19,000	Gerontology	551 232	D13	Xavier
200450	Earl	290955	160778	26,000	Adult	896 467	D21	Zalkin
200500	Flutter	160461	150783	22,800	Perinatal	756 498	D14	Gerald
200550	Forster	201170	120793	21,000	Surgical	692 441	D13	Xavier
200600	Glover	041074	270798	23,500	Adult	521 886	D14	Gerald
200650	Gray	160875	180799	21,800	Perinatal	821 490	D11	Baker
200700	Hatcher	271065	210788	24,600	Psychiatry	941 724	D22	Beaver
200750	Heaton	230859	240783	30,580	Gerontology	578 452	D13	Xavier
200800	Hurst	061076	200799	21,341	Midwifery	790 442	D11	Baker
200850	Jenkins	020568	120793	23,670	Paediatrics	674 251	D14	Gerald
200900	Jervis	140973	190796	24,656	Geriatrics	322 890	D19	Usher
200950	Jessop	281072	110795	23,510	Surgical	983 221	D23	Hagger
201000	Kemp	131276	280798	24,970	Adult	452 775	D14	Gerald
201050	Knight	210677	220700	20,360	Perinatal	674 218	D11	Baker
201100	Lucas	260758	260784	32,000	Surgical	540 328	D13	Xavier
201150	Ludgate	080568	220790	24,800	Child	674 543	D23	Hagger
201200	Malcolm	061071	140793	21,723	Gerontology	353 769	D13	Xavier
201250	Mason	130173	260796	22,920	Adult	253 762	D21	Zalkin
201300	Neaves	160857	300779	33,200	Midwifery	663 741	D11	Baker
201350	Newman	041271	170794	22,700	Psychiatry	287 441	D22	Beaver
201400	Oscar	060475	220798	21,200	Perinatal	531 442	D11	Baker
201450	Oudes	230975	260798	22,100	Paediatrics	965 432	D14	Gerald
201500	Peace	221073	120795	22,000	Surgical	471 663	D23	Hagger
201550	Petrie	070564	230787	31,000	Geriatrics	421 875	D19	Usher
201600	Reuter	200771	180793	24,350	Adult	854 331	D14	Gerald
201650	Russell	120973	250795	23,800	Child	247 890	D23	Hagger
201700	Scales	191270	140793	23,900	Midwifery	421 776	D11	Baker
201750	Speller	130877	120798	22,000	Surgical	993 567	D23	Hagger
201800	Staples	220269	170793	21,788	Psychiatry	651 774	D22	Beaver
201850	Swanton	010775	120797	18,922	Child	531 442	D23	Hagger
201900	Tam	200181	220701	16,300	Child	661 476	D23	Hagger
201950	Tardif	220768	140790	27,600	Geriatrics	552 659	D19	Usher
202000	Taylor	100377	210799	21,677	Gerontology	542 774	D13	Xavier
202050	Turpin	241070	260792	24,344	Adult	471 922	D14	Gerald
202100	Vallis	140674	270796	21,300	Psychiatry	367 884	D22	Beaver
202150	Veitch	110674	120796	22,100	Perinatal	422 7791	D11	Baker
202200	Vernon	181072	160794	23,838	Paediatrics	472 1562	D14	Gerald
202250	Viner	080665	120787	30,700	Perinatal	674 228	D11	Baker
202300	Walker	110972	230795	23,650	Surgical	571 224	D23	Hagger
202350	Weaver	240363	170785	27,500	Gerontology	641 833	D13	Xavier
202400	Watson	110577	280799	22,150	Geriatrics	431 688	D19	Usher
202450	Waugh	310873	200795	22,200	Child	598 331	D23	Hagger

Figure A.2: The Nurse relation.

P_SSN	P_Name	P_DOB	P_Status	P_Tel.	INS_Name	PHY_Name	EC_Name	Patient #
100000	Aaron	120955	Outpatient	453 6645	BlueCross	Piper	Bolwell	124
100050	Baker	240351	Inpatient	324 8030	Geico	Stanley	Hussey	133
100100	Cantor	210265	Outpatient	463 6211	CareFirst	Albert	Lewis	212
100150	Davies	090552	Inpatient	393 2222	Humana	Farah	West	214
100200	Earl	160961	Outpatient	727 3323	Kaiser	Duncan	Dickson	109
100250	Fagin	271153	Inpatient	832 4357	Cigna	Archer	Butler	173
100300	Garrod	210464	Inpatient	442 4320	Aetna	Wood	Lloyd	663
100350	Hall	240360	Outpatient	442 9200	CapitalCare	Wolfgang	Ford	256
100400	Ian	010755	Inpatient	332 2630	PrimeHealth	Brown	Fraser	389
100450	James	131066	Inpatient	736 1775	Trigon	Higgins	Andrews	765
100500	Kain	230848	Outpatient	842 7621	Concordia	Vinall	Angus	932
100550	Laws	050150	Inpatient	338 2969	Potomac	Piper	Duncan	512
100600	Martin	120857	Outpatient	232 2232	AlliedHealth	Booker	Beasley	902
100650	Nash	280363	Inpatient	244 0800	Metropolitan	Weaver	Davies	767
100700	Oakley	041268	Outpatient	362 6120	Prudential	Booth	Morris	550
100750	Parker	180464	Inpatient	363 5803	Providence	Collins	Rider	700
100800	Qassem	291267	Inpatient	291 9341	Humana	Conolly	Bright	212
100850	Ratner	230253	Outpatient	723 6670	Concordia	Fletcher	Crane	908
100900	Sachs	020959	Inpatient	635 6563	Potomac	Rahman	Crompton	113
100950	Taylor	221148	Inpatient	398 5101	Metropolitan	Freeman	Wilson	141
101000	Usher	180456	Outpatient	724 9702	Trigon	Gunn	Gates	155
101050	Valdez	170365	Inpatient	582 6360	PrimeHealth	Sumter	Gardner	673
101100	Walker	111063	Outpatient	889 9745	Kaiser	Pierce	Hancock	326
101150	Xavier	290860	Inpatient	562 1188	Cigna	Johnson	Gould	182
101200	Yates	170951	Inpatient	234 1723	BlueCross	Huskins	Field	258
101250	Zalkin	081260	Outpatient	872 9200	Geico	Foster	Neville	193
101300	Able	300152	Inpatient	363 8060	Kaiser	Keenan	Blackwell	284
101350	Beaver	140363	Outpatient	466 4270	Potomac	Warren	Hughes	337
101400	Ceasar	170956	Inpatient	984 1800	Humana	Abbott	Harvey	792
101450	Dean	230448	Inpatient	652 9188	Metropolitan	Warwick	Potter	375
101500	Ebert	140658	Outpatient	547 2355	Concordia	Walker	Green	161
101550	Feeney	121061	Inpatient	466 8312	Prudential	Clark	Allen	982
101600	Gerald	160357	Outpatient	383 7451	AlliedHealth	Preston	Nash	158
101650	Henry	180960	Inpatient	331 7100	BlueCross	Lawrence	Ince	210
101700	Ingram	030262	Inpatient	328 9449	Humana	Wesley	Wood	112
101750	Jessup	090964	Outpatient	628 2209	Cigna	Weiss	Alderton	637
101800	Kerry	130667	Inpatient	237 1788	Geico	Clifford	Scott	818
101850	Leach	050565	Inpatient	635 5299	Potomac	Sutton	Cox	166
101900	Meadow	040360	Inpatient	828 4908	Trigon	Gupta	Gibbons	222
101950	Nelson	131252	Outpatient	955 9600	Concordia	Peters	Hall	313
102000	Oberst	201161	Inpatient	835 2041	CapitalCare	Shepperd	Fowler	111
102050	Peters	010858	Outpatient	508 6059	PrimeHealth	Schneider	Wright	180
102100	Quaye	260554	Inpatient	622 1904	Geico	Bruce	Roper	397
102150	Reaves	080750	Outpatient	554 8489	Metropolitan	Yates	Turner	788
102200	Scales	171251	Inpatient	289 9838	BlueCross	Mercer	Hamilton	285
102250	Temple	031259	Inpatient	829 4458	Kaiser	Fulford	Drake	135
102300	Ullman	021048	Outpatient	635 7025	Humana	Rosenthal	Carter	266
102350	Verdi	250961	Inpatient	965 0675	Providence	Brendan	Sullivan	377
102400	Wells	161163	Inpatient	463 0950	Trigon	Rothman	Facey	381
102450	Xhing	281258	Outpatient	546 3311	Aetna	Griffin	Murphy	214

Figure A.3: A subset of the Patient relation.

Appendix B

Lattice Routine Algorithm

Require: List of elements of type LABEL
Ensure: List of elements constitute a lattice

```
LABEL Element[64], SystemHigh, SystemLow, SecurityLabel, Res;
INT IsIndex = 0, MaxElem = 0, ArrayIndex = 0, track = 0, count = 0;
BOOL boolDominate[64][64]= FALSE, ArrayDominate[64] = FALSE, ArrayDominant[64] = FALSE;

while SecurityLabel $\neq$ '.' do
  Input SecurityLabel
  Element[ArrayIndex] = SecurityLabel; MaxElem++; ArrayIndex++;
end while
for (ArrayIndex = 0; Arrayindex < MaxElem; Arrayindex++) do
  boolDominate[ArrayIndex][ArrayIndex] = TRUE;
end for
for (ArrayIndex = 0; ArrayIndex < MaxElem; ArrayIndex++) do
  for (IsIndex = 0; IsIndex < MaxElem; IsIndex++ ) do
    Output Does Element[ArrayIndex] Dominates Element[IsIndex]
    Input Res;
    if (Res = "Yes") then
      boolDominate[ArrayIndex][IsIndex] = TRUE;
    end if
  end for
end for
for (ArrayIndex = 0; Arrayindex < MaxElem; ArrayIndex++ ) do
  for (IsIndex = Arrayindex + 1; IsIndex < MaxElem; IsIndex++ ) do
    for (m = 0; m < MaxElem; m++) do
      if (boolDominate[ArrayIndex][m] = TRUE and boolDominate[IsIndex][m] = TRUE)
      then
    ArrayDominate[m] = TRUE;
  end if
  if (boolDominate[m][ArrayIndex] = TRUE and boolDominate[m][IsIndex] = TRUE)
  then
    ArrayDominant[m] = TRUE;
  end if
end for
if (TRUE $\notin$ ArrayDominate[0..MaxElem] or TRUE $\notin$ ArrayDominant[0..MaxElem])
then
  Failure: This set of labels is not a security lattice; STOP
end if
for (i = 0; i < MaxElem; i++) do
  for (j = i + 1; j < MaxElem; j++) do
    if (ArrayDominate[i] = TRUE and ArrayDominate[j] = TRUE) then
      if (boolDominate[i][j] = TRUE) then
        ArrayDominate[j] = FALSE;
      else if (boolDominate[j][i] = TRUE) then
        ArrayDominate[i] = FALSE;
      end if
    end if
    if (ArrayDominant[i] = TRUE and ArrayDominant[j] = TRUE) then
      if (boolDominate[i][j] = TRUE) then
        ArrayDominant[i] = FALSE;
      else if (boolDominant[j][i] = TRUE) then
        ArrayDominant[j] = FALSE;
      end if
    end if
  end for
end for
for (num = 0; num < MaxElem; num++) do
  if (ArrayDominate[num] = TRUE) then
    count++;
  end if
  if (ArrayDominant[num] = TRUE) then
    track++;
  end if
end for
if (count $\neq$ 1 or track $\neq$ 1) then
```

```
      Failure: This set of labels is not a security lattice; STOP
    end if
    ArrayDominate[0..MaxElem] = FALSE; ArrayDominant[0..MaxElem] = FALSE;
    track = 0; count = 0;
  end for
end for
Success: This set of labels forms a lattice
```

Bibliography

[1] Multilevel data management security. Technical report, Air Force Studies Board Committee on Multilevel Data Management Security, Washington, DC, March 1983.

[2] M.D. Abrams, S. Jajodia, and H. Podell, editors. *Information Security: An Integrated Collection of Essays.* IEEE Computer Society Press, Los Alamitos, California, 1998.

[3] J.P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51-1, U.S. Air Force Electronic Systems Division, Bedford, Massachusetts, October 1972.

[4] J.P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51-2, U.S. Air Force Electronic Systems Division, Bedford, Massachusetts, October 1972.

[5] ANSI. *Database Language SQL.* American National Standards Institute, Washington, DC, x3.135-1992 edition, 1992.

[6] V. Atluri, S. Jajodia, and B. George. *Multilevel Secure Transaction Processing.* Kluwer Academic Publishers, Boston, Massachusetts, 1999.

[7] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report ESD-TR-73-278-1, MITRE Corporation, Bedford, Massachusetts, November 1973.

[8] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report ESD-TR-73-278-2, MITRE Corporation, Bedford, Massachusetts, November 1973.

[9] P.A. Bernstein, V. Hadzilacos, N. Goodman, and V. Radzilacos. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, Massachusetts, 1987.

[10] K.J Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, U.S. Air Force Electronic Systems Division, Bedford, Massachusetts, April 1977.

[11] H. Bidgoli and R. Azarmsa. Computer Security: New managerial concern for the 1980s and beyond. *Journal of Systems Management*, 2:21–27, November 1989.

[12] D. Bitton, D.J. DeWitt, and C. Turbyfill. Benchmarking database systems: A systematic approach. In *Proceedings, 9th International Conference on Very Large Databases*, pages 8–19, Florence, Italy, October 1983. Morgan-Kaufmann.

[13] D.F.C. Brewer and M.J. Nash. The Chinese Wall security policy. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 206–214, Oakland, California, May 1989. IEEE Computer Society Press.

[14] U. Bussolati, M.G. Fugini, and G. Martella. A conceptual framework for security systems: The action-entity model. In *Proceedings, 9th IFIP World Conference*, pages 127–132, Paris, France, September 1983. IFIP Press.

[15] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security.* Addison-Wesley, Essex, England, 1994.

[16] S. Ceri, B. Navathe, and G. Wiederhold. Distribution design of logical database schemas. *IEEE Transactions on Software Engineering*, 9(4):487–504, November 1983.

[17] S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. In *Proceedings, ACM SIGMOD International Conference on Management of Data*, pages 128–136, Orlando, Florida, June 1982. ACM Press.

[18] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill, New York, New York, 1984.

[19] S.K. Chang and W.H. Cheng. A methodology for structured database decomposition. *IEEE Transactions on Software Engineering*, 6(2):205–218, March 1980.

[20] F. Chen and R. Sandhu. The semantics and expressive power of the MLR data model. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 128–142, Oakland, California, April 1995. IEEE Computer Society Press.

[21] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, California, April 1987. IEEE Computer Society Press.

[22] O. Costich, J. McLean, and J. McDermott. Confidentiality in a replicated architecture trusted database system: A formal model. In *Proceedings of the Computer Security Foundations Workshop VII*, pages 60–65, Franconia, New Hampshire, June 1994. IEEE Computer Society Press.

[23] C. Dalton and T.H. Choo. An operating system approach to securing e-services. *Communications of the ACM*, 44(2):58–64, February 2001.

[24] C.J. Date. *An Introduction to Database Systems.* Addison-Wesley, Reading, Massachussetts, 8th edition, 2003.

[25] D.E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[26] D.E. Denning. Commutative filters for reducing inference threats in multilevel database systems. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 134–146, Oakland, California, April 1985. IEEE Computer Society Press.

[27] D.E. Denning, S.G. Akl, M. Heckman, T.F. Lunt, M. Morgenstern, P.G. Neumann, and R.R. Schell. Views for multilevel database security. *IEEE Transactions on Software Engineering*, 13(2):129–140, February 1987.

[28] D.E. Denning, T.F. Lunt, R.R. Schell, W. Shockley, and M. Heckman. The SeaView security model. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 218–233, Oakland, California, April 1988. IEEE Computer Society Press.

[29] D.J. DeWitt, S. Ghandeharizadeh, and D. Schneider. A performance analysis of the Gamma database machine. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 17(3):350–360, September 1988.

[30] DoD. Department of defense privacy program. Directive 5400.11, U.S. Department of Defense, Washington, DC, June 1982.

[31] DoD. Security requirements for automated information systems (AISs). Directive 5200.28, U.S. Department of Defense, Washington, DC, March 1988.

[32] D. Downs and G.J. Popek. A kernel design for a secure database management system. In *Proceedings, 3rd International Conference on Very Large Databases*, pages 507–514, Tokyo, Japan, October 1977. IEEE Computer Society Press.

[33] P. Dwyer, G. Jelatis, and B.M. Thuraisingham. Multilevel security in database management systems. *Computers and Security*, 6(3):252–260, June 1987.

[34] C. Dye. *Oracle Distributed Systems.* O'Reilly & Associates Inc., Sebastopol, California, 1999.

[35] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book.* Prentice Hall, Upper Saddle River, New Jersey, 2002.

[36] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 2, pages 1–39, River Edge, New Jersey, 1992. World Scientific Publishing Company.

[37] M. Gasser. *Building a Secure Computer System.* Van Nostrand Reinhold, New York, 1988.

[38] D. Gollmann. *Computer Security.* John Wiley & Sons, Chichester, England, 1999.

[39] R. Graubart. The integrity-lock approach to secure database management. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 62–74, Oakland, California, April 1984. IEEE Computer Society Press.

[40] J. Gray, editor. *The Benchmark Handbook for Database and Transaction Systems.* Morgan-Kaufmann, San Francisco, California, 2nd edition, 1993.

[41] J. Gray and A. Reuter. *Transaction Processing: Concepts and techniques.* Morgan-Kaufmann, San Francisco, California, 1993.

[42] J.T. Haigh, R.C. O'Brien, and D.J. Thomasen. The LDV secure relational DBMS model. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and prospects*, pages 265–280. Elsevier Science, North-Holland, January 1991.

[43] Honeywell. Secure distributed data views — security policy extensions. Technical Report A002, Honeywell Systems Research Center and Corporate Systems Development Division, St. Anthony, Minnesota, April 1987.

[44] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for experimental design, measurement, simulation, and modeling.* John Wiley & Sons, Hoboken, New Jersey, 1991.

[45] S. Jajodia and R. Mukkamala. Effects of SeaView decomposition of multilevel relations on database performance. In S. Jajodia and C.E. Landwehr, editors, *Database Security V: Status and prospects*, pages 203–225. Elsevier Science, North-Holland, January 1992.

[46] S. Jajodia and R. Sandhu. Polyinstantiation integrity in multilevel relations. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 104–115, Oakland, California, May 1990. IEEE Computer Society Press.

[47] S. Jajodia and R. Sandhu. A novel decomposition of multilevel relations into single-level relations. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 300–313, Oakland, California, May 1991. IEEE Computer Society Press.

[48] S. Jajodia and R. Sandhu. Towards a multilevel secure relational data model. In J. Clifford and R. King, editors, *Proceedings, ACM SIGMOD International Conference on Management of Data*, pages 50–59, Denver, Colorado, May 1991. ACM Press.

[49] A.K. Jones, R.J. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *Proceedings, 17th IEEE Symposium on Foundations of Computer Science*, pages 33–41, Houston, Texas, October 1976. IEEE Computer Society Press.

[50] N. Jukic, S. Vrbsky, A. Parrish, B. Dixon, and B. Jukic. A belief-consistent multilevel secure relational data model. *Information Systems Journal*, 24(5):377–400, July 1999.

[51] M.H. Kang, A.P. Moore, and I.S. Moskowitz. Design and Assurance Strategy for the NRL Pump. *IEEE Computer*, 31(4):56–64, April 1998.

[52] T.F. Keefe, M.B. Thuraisingham, and W.T. Tsai. Secure query-processing strategies. *IEEE Computer*, 22(3):63–70, March 1989.

[53] K.Henry. Legacy multilevel secure database management systems: The future. Technical Report DARPA-DSO-SR-15, Defense Advanced Research Projects Agency, Arlington, Virginia, May 2000.

[54] W. Kim, D. Reiner, and D. Batory, editors. *Query Processing in Database Systems.* Springer-Verlag, London, England, 1985.

[55] R. Knapman, J. Bryant, and C. Martin. The Stargres release 3.2: Software structure. Technical Report APL-CAIR-328-3, Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland, April 2000.

[56] R. Knapman, M. Furst, D. Shtengel, and L. Freeman. The Stargres release 3.2: StarQL. Technical Report APL-CAIR-328-4, Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland, October 2000.

[57] B. Lampson. Protection. *ACM Operating System Reviews*, 8(1):18–24, January 1974.

[58] C.E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[59] M. Levene and G. Loizou. *A Guided Tour of Relational Databases and Beyond.* Springer-Verlag, London, England, 1999.

[60] T. Lunt, editor. *Research Directions in Database Security.* Springer-Verlag, New York, New York, 1992.

[61] T.F. Lunt, R.R. Schell, W. Shockley, M. Heckman, and D. Warren. A near-term design for the SeaView multilevel database system. In *Proceedings, IEEE Symposium on Security and Privacy*, pages 234–244, Oakland, California, June 1988. IEEE Computer Society Press.

[62] J. McLean. A comment on the 'Basic Security Theorem' of Bell and LaPadula. *Information Processing Letters*, 20(2):67–70, February 1985.

[63] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, December 1989.

[64] NCSC. Department of defense trusted computer system evaluation criteria. Report DoD 5200.28-STD, National Computer Security Centre, Washington, DC, December 1985. Orange Book edition.

[65] NCSC. Trusted network interpretation of the trusted computer systems evaluation criteria (TCSEC). Report NCSC-TG-005, National Computer Security Centre, Washington, DC, July 1987.

[66] NCSC. Trusted database management system interpretation of the trusted computer system evaluation criteria (TCSEC). Report NCSC-TG-021, National Computer Security Centre, Washington, DC, April 1991.

[67] NIST. Common criteria for information technology security evaluation. Report 2.1, National Institute of Standards and Technology, Washington, DC, August 1999.

[68] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 1999.

[69] G. Pfister. *In Search of Clusters: The ongoing battle in lowly parallel computing.* Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 1998.

[70] C.P. Pfleeger and S.L. Pfleeger. *Security in Computing.* Prentice Hall, Englewood Cliffs, New Jersey, 3rd edition, 2003.

[71] J. Rushby and B. Randell. A distributed secure system. *IEEE Computer*, 16(7):55–67, July 1983.

[72] D. Russell and G.T. Gangemi. *Computer Security Basics.* O'Reilly & Associates Inc., Sebastopol, California, 1991.

[73] SCC. Locked workstation program LWS expanded environment study report. Technical Report SCS-1-96, Secure computing corporation, San Jose, California, January 1996.

[74] R.R. Schell, T.F. Tao, and M. Heckman. Designing the gemsos security kernel for security and performance. In *Proceedings, 8th DoD/NBS Computer Security Conference*, pages 108–119, Gaithersburg, Maryland, May 1985. IEEE Computer Society Press.

[75] K. Smith and M. Winslett. Entity modelling in the MLS relational model. In Li-Yan Yuan, editor, *Proceedings, 18th International Conference on Very Large Databases*, pages 199–210, Vancouver, Canada, August 1992. Morgan-Kaufmann.

[76] M. Stonebraker and E. Wong. Access control in a relational database management system by query modification. In *Proceedings, 1974 ACM Annual Conference*, pages 180–186, New York, May 1974. ACM Press.

[77] B. Thuraisingham and A. Kamon. Query processing in a trusted database management system: Design and performance study. Technical Report MTP-292, MITRE Corporation, Bedford, Massachusetts, June 1990.

[78] B. Thuraisingham and A. Kamon. Secure query processing in distributed database management systems: Design and performance studies. In *Proceedings, 6th Annual Computer Security Applications Conference*, pages 88–102, Tucson, Arizona, December 1990.

[79] TPC. TPC Benchmark A. Standard Specification Report TPC-A-2.0-94, Transaction Processing Performance Council, San Francisco, California, June 1994.

[80] TPC. TPC Benchmark B. Standard Specification Report TPC-B-2.0-94, Transaction Processing Performance Council, San Francisco, California, June 1994.

[81] TPC. TPC Benchmark C. Standard Specification Report TPC-C-5.2-03, Transaction Processing Performance Council, San Francisco, California, December 2003.

[82] C. Turbyfill, C. Orji, and D. Bitton. AS^3AP: A comparative relational database benchmark. In *Proceedings, 34th IEEE Computer Society International Conference*, pages 560–564, San Francisco, California, February 1989. IEEE Computer Society Press.

[83] J. Wilson. A security policy for an A1 DBMS (a trusted subject). In *Proceedings, IEEE Symposium on Security and Privacy*, pages 116–125, Oakland, California, May 1989. IEEE Computer Society Press.

[84] S.R. Wiseman. Purple Penelope: Extending the security of Windows NT. Technical Report RSRE-97224, Defence Research Agency, Malvern, England, February 1997.

[85] A.W. Wood, S.R. Lewis, and S.R. Wiseman. The SWORD multilevel secure DBMS. Technical Report RSRE-92005, Defence Research Agency, Malvern, England, May 1992.

Index