SQL Extensions for Security Assertions

Paul STACHOUR * and Bhavani THURAISINGHAM **
Honeywell Inc., Minneapolis, Minnesota, USA

We describe the extensions to the relational query language SQL that are necessary to support multilevel security in database management systems. In a multilevel secure database with data at a variety of sensitivity levels, the data classifications are provided by security assertions. Consequently, the data definition language and the data manipulation language should provide support for incorporating these assertions.

Keywords: Multilevel Secure Database Management Systems, SQL extensions, Security assertions, Relational data model, Schema, Classification levels.

Paul D. Stachour, a Principal Research Scientist at Secure Computing Technology Corporation, has a PhD in Computer Science from the University of Waterloo, Waterloo, Ontario. His primary research areas are Secure Multilevel Applications, Software Component Design, Program Design Methodologies, and Software Development Styles using Ada.

Dr. Stachour previously worked at Honeywell's Corporate Systems Development Division and Honeywell's Computer Sciences Center. Prior to that he worked on the PL/S Compiler at IBM's Systems Development Division and on a US Army Curriculum Management system at the Army Chemical School.

He is a coauthor of the book *Ada, A Programmer's Guide, with Microcomputer Examples*.

Bhavani Thuraisingham is a lead engineer at the MITRE Corporation. Her research interests include database security and the applications of mathematical logic in computer science. Previously she was at Honeywell Inc. where she was involved in the design of Lock Data Views. She was also an adjunct professor and member of the graduate faculty in the Department of Computer Science at the University of Minnesota.

Dr. Thuraisingham received her Ph.D. degree in recursive functions and computability theory from the University of Wales, Swansea, United Kingdom. She has published more than 25 journal papers in database security, computability theory, AI and distributed processing.

* Paul Stachour is at present with Secure Computing Technology Corporation, Arden Hills, Minnesota, USA.
** Bhavani Thuraisingham is at present with The MITRE Corporation, Bedford, Massachusetts, USA.

1. Introduction

Within the department of defense (DOD), the number of computerized databases containing classified or otherwise sensitive data is increasing rapidly. Access to these databases must be restricted and controlled to limit the unauthorized disclosure, or malicious modification, of data contained in them. Present DBMSs do not provide adequate mechanisms to support such control. Penetration studies have clearly shown that the mechanisms provided even by "security enhanced" database systems can be bypassed, often due to fundamental flaws in the systems which host the DBMS. This has led to a reliance on a number of techniques for isolating sensitive database information. These include physical protection, "system high" operation, and use of manual techniques for data sharing.

TCBs such as Honeywell's LOCK [4], have been designed to provide this type of operating system orientation. The LOCK security policy consists of a discretionary security policy and a mandatory security policy. The discretionary security policy enforces the need to know structures, while the mandatory security policy provides a multilevel control policy. The multilevel control policy is a non-interference policy which addresses both accesses to data and the flow of information in the system.

A DBMS presents a more difficult security problem than that dealt with by current TCBs with their operating system orientation. This results from the ability of the DBMS to preserve or even enhance the information value of the data it contains. This is possible because it captures information in addition to the raw data values themselves through the incorporation of knowledge about the types of data and relationships among the data elements. A DBMS also allows for the creation of new data and relationships through the application of complex functions to the data. Be-

1 Part of the Introduction has appeared in [11], © 1990 IEEE. Written permission has been obtained by the authors from IEEE to use this part.
cause of these capabilities, one is forced to consider a number of factors beyond those normally addressed when dealing with operating system security. These include the impact of data context, aggregation, and inference potential (see [11] for a more complete description of the issues involved).

Honeywell's LDV (LOCK Data Views) system [11], which is a multilevel secure relational database management system (MLS/DBMS), addresses the above problems by allowing individuals possessing a range of clearances to create, share and manipulate databases containing information spanning multiple sensitivity levels. In LDV, the relational query language Structured Query Language (SQL) [3,5], is enhanced with constructs for formulating security assertions. These security assertions serve to imply sensitivity labels (or security levels) for all atomic values, contexts and aggregations in a database. The labeled data are partitioned across security levels, assigned to containers with dominating security markings or levels, and may only flow upward in level unless authorized otherwise. The design of LDV which describes the query, update and metadata management operations have been described in [9,11]. The focus of this paper are the SQL extensions that are required to support multilevel security. SQL was selected as the data definition and data manipulation language because it has been adopted as a standard by the American National Standards Institute (ANSI) [2].

The organization of this paper is as follows. In Section 2 we describe the security assertions which enforce data classifications. In Section 3 we describe the extensions that we are proposing that should be made to SQL in order to support some of the security assertions. A parallel effort is also being made to design a multilevel secure relational database system. This effort is also proposing SQL extensions to support multilevel security [10].

2. Security Assertions

There are two types of security assertions that may be enforced. One is mandatory assertions which assign security levels to data based on content, context and aggregation among others. The other is discretionary assertions which authorize a user or a user group to access certain database entities. Discretionary assertions are also known as access control rules [12]. Authorizations could also be given to users based on the content of the data and the context in which it is displayed. This paper will be limited to a discussion on mandatory assertions only. It should be noted however, similar SQL extensions can be proposed for various types of discretionary assertions.

Mandatory assertions, which are also called security constraints or classification constraints, provide a powerful, versatile, classification policy which can assign security levels, also called classification levels, to data based on content, context, aggregation and time among others [7]. For exam-
ple, a simple constraint classifies a database entity regardless of the database content. It could also classify an entire database. Content-based constraints classify a database entity depending on its content or the content of some other entity. Context-based constraints classify the relationships between database entities. Aggregate constraints classify collection of entities taken together. Time based constraints classify entities dynamically. Examples of these constraints are given below using a relational database [6] which consists of a relation EMP with attributes SS#, Name and Salary.

Simple constraint: EMP is secret
Content-based constraint: Name in EMP is secret if Salary > 50K
Content-based constraint: Name and Salary values taken together is secret.
Aggregate constraint: More than 10 names in EMP is secret.
Time constraint: After 1/1/90, the names in EMP will be unclassified.

3. LDV Usage

This section describes LDV from the users point of view. Specifically the LDV data model, data independence issues and the essential points of the language used as the interface to the system are described. As stated earlier, this language is the SQL relational language with appropriate extensions for the security assertions.

3.1. LDV Data Model

LDV implements the relational model of data [6]. A relation is a tabular representation of information consisting of a number of columns (attributes) and rows (tuples) as shown in Fig. 1. Within each row, a column may have only a single value (element). The set of all possible values that a column may have is known as its domain. Each row is uniquely identified by the values of one or more columns, collectively known as the primary key. Two relations R and S with keys a and d respectively are shown in Fig. 2.

3.2. Data Independence

Data independence for the application program is an important requirement of database systems. There are two types of data independence: logical data independence and physical data independence. Logical data independence is the ability to support new applications requiring new or existing data without changing existing programs. Physical data independence is the ability to change physical storage characteristics without changing any of the application programs.

Data independence is achieved by describing data at three different levels of abstraction and providing mappings between the levels as shown in Fig. 3. Corresponding to each level is a description of the data at that level, called the schema. On the application side, the external schemas describe each user's view of the data model and define the external data models. On the database side, the internal schema describes how data are stored. In between is the conceptual schema that defines the conceptual data model, which is an abstract representation of the data in the database and its semantics. At this level the data are organized into base relations.

It is the ability to provide independent external data views for individual users (or applications), without having to modify the conceptual and internal schemas, that was an early motivating factor for this work. It was observed by several individuals that because data views can define arbitrary sets of stored and derived data, they could provide a means of addressing the problems of security for a dynamic database. The output of the 1982 Woods Hole Summer Study on Multi-level Database Management Security [1] reaffirmed the promise of this concept. We have found many of the concepts underlying the view mechanisms to be of benefit in our system. However, the specific notion of data views in use within the database community is not an explicit part of our security policy implementation.
3.3. LDV Language

The DBMS must provide a language for use in describing the various schemas and for retrieving and maintaining the data in the database. This language allows users to deal with the database in abstract terms: to be concerned with what must be done and not how it must be done. Three distinct user roles are considered; ordinary users, database administrators (DBAs) and database system security officers (DBSSOs). Ordinary users perform query, insert, delete and update operations on the data. DBAs maintain the metadata, i.e. the data that describes the database data. DBSSOs maintain the fundamental classification rules.

The LDV language consists of two parts; a data definition language (DDL), and data manipulation language (DML). The DDL is used by the DBA and the DBSSO to describe the data, and the DML is used by ordinary users to retrieve and maintain the data. The DDL and DML for this design are based on the SQL language being proposed by the American National Standards Institute (ANSI) [2].

We have extended the DDL to allow for the specification of the classification constraints, primary keys and the derivation of values of attributes of one tuple from those of other tuples (see [11] for a complete description of the derivation of values). The DML has been extended with
a time-oriented construct that refers to points in
time, and a level-oriented construct that refers to
the classification levels of data.

The definition of the database language is pre-
sented in terms of user roles. The DBA uses the
DDL to define external, conceptual and internal
schemas, the DBSSO uses the DDL to define
security constraints, and the ordinary users use the
DML to retrieve and maintain data. The exten-
sions to the ANSI SQL version, called SQL2, are
indicated in boldface and are discussed below. A
discussion on ANSI SQL2 language can be found
in [2].

The SQL interface is written in a command line
oriented form. The syntactic notation follows that
of [2], which is Bakus-Normal Form (BNF), with
the following extensions.

Square brackets ([ ]) indicate optional state-
ments
Ellipses (...) indicate elements that may be re-
peated one or more times
Braces ({ }) group sequence of elements.

3.3.1. DBA Language

3.3.1.1 External Schema The DBA Creates a view
using the create statement:

\[
\text{CREATE VIEW} \quad \text{(table name)} \quad \text{AS} \quad \text{(query expression)}
\]

\[
\text{WITH CHECK OPTION}
\]

\[
\text{(table name)} :=
\]

\[
\text{(qualified name)}
\]

\[
\text{|MODULE. (local table name)}
\]

\[
\text{(local table name)} :=
\]

\[
\text{(identifier)}
\]

\[
\text{(view column list)} :=
\]

\[
\text{(column name) [{, (column name)} ...]}
\]

\[
\text{(column name)} :=
\]

\[
\text{(identifier)}
\]

\[
\text{(query expression)} :=
\]

\[
\text{(query term)}
\]

\[
\text{|(query expression) UNION [ALL]}
\]

\[
\text{|(corresponding specification)} \quad \text{(query term)}
\]

\[
\text{|(query expression) EXCEPT [ALL]}
\]

\[
\text{|(corresponding specification)} \quad \text{(query term)}
\]

\[
\text{(corresponding specification)} :=
\]

\[
\text{CORRESPONDING [BY (corresponding column list)]}
\]

\[
\text{(corresponding column list)} :=
\]

\[
\text{(column name) [{, (column name)} ...]}
\]

\[
\text{(query term)} :=
\]

\[
\text{(query primary)}
\]

\[
\text{|(query term) INTERSECT [ALL]}
\]

\[
\text{(corresponding specification)} \quad \text{(query primary)}
\]

\[
\text{(query primary)} :=
\]

\[
\text{(simple table)}
\]

\[
\text{|(joined table)}
\]

\[
\text{|(query expression)}
\]

\[
\text{(simple table)} :=
\]

\[
\text{(query specification)}
\]

\[
\text{|(table value expression)}
\]

\[
\text{|(table value expression)} :=
\]

\[
\text{TABLE [VALUES]}
\]

\[
\text{(row value expression) [{, (row value expression)} ...]}
\]

\[
\text{(row value expression)} :=
\]

\[
\text{(row value expression element)}
\]

\[
\text{|(row value expression list)}
\]

\[
\text{|(row subquery)}
\]

\[
\text{(row value expression element)} :=
\]

\[
\text{(value expression)}
\]

\[
\text{|(null specification)}
\]

\[
\text{(null specification)} :=
\]

\[
\text{NULL}
\]

\[
\text{(row value expression list)} :=
\]

\[
\text{(row value expression element) [{, (row value expression element)} ...]}
\]

\[
\text{(row subquery)} :=
\]

\[
\text{(subquery)}
\]

\[
\text{(subquery) :=
\]

\[
\text{(query expression)}
\]

\[
\text{|(joined table)} :=
\]

\[
\text{(table reference)}
\]

\[
\text{NATURAL [(join type)] JOIN}
\]

\[
\text{BY (join column list)}
\]

\[
\text{(table reference)}
\]

\[
\text{|(table reference)}
\]

\[
\text{|(table reference)}
\]

\[
\text{JOIN (table reference)}
\]

\[
\text{ON (join condition)}
\]

\[
\text{(join type)} :=
\]

\[
\text{INNER}
\]

\[
\text{LEFT}
\]

\[
\text{RIGHT}
\]

\[
\text{FULL}
\]

\[
\text{UNION}
\]

\[
\text{(join column list)} :=
\]

\[
\text{(column name) [{, (column name)} ...]}
\]

\[
\text{(join condition)} :=
\]

\[
\text{(search condition)}
\]

The definitions of (query specification), (table
reference), (search condition), and (value expres-
sion) will be given in Section 3.3.3.1. For the
definition of (identifier) we refer to [2].

The DBA deletes a view using the drop state-
ment:

\[
\text{DROP VIEW} \quad \text{(table name)} \quad \text{[CASCADE]}
\]

The DBA defines discretionary access rights using
the grant statement:

\[
\text{GRANT (privileges) ON (object name)}
\]

\[
\text{TO (grantee) [{, (grantee)} ...]}
\]

\[
\text{[WITH GRANT OPTION]}
\]

\[
\text{(privileges) :=
\]

\[
\text{ALL PRIVILEGES}
\]

\[
\text{|(action) [{, (action)} ...]}
\]
\(\langle\text{action}\rangle\) ::= 
  \text{SELECT} 
  | \text{DELETE} 
  | \text{INSERT} [[\langle\text{grant column list}\rangle]] 
  | \text{UPDATE} [[\langle\text{grant column list}\rangle]] 
  | \text{REFERENCES} [[\langle\text{grant column list}\rangle]] 
  | \text{USAGE} 
\langle\text{grant column list}\rangle ::= 
  \langle\text{column name}\rangle [[, \langle\text{column name}\rangle]…] 
\langle\text{grantee}\rangle ::= 
  \text{PUBLIC} 
  | [\langle\text{user authorization identifier}\rangle] 
\langle\text{object name}\rangle ::= 
  \langle\text{table name}\rangle 
  | \langle\text{domain name}\rangle 
\langle\text{domain name}\rangle ::= 
  \langle\text{qualified name}\rangle 

For the definitions of \(\langle\text{qualified name}\rangle\) and \(\langle\text{user authorization identifier}\rangle\) we refer to [2].

The DBA revokes discretionary access rights using the revoke statement.
\(\langle\text{revoke statement}\rangle\) ::= 
\text{REVOKE} \langle\text{GRANT OPTION FOR}\rangle \langle\text{privileges}\rangle 
ON \langle\text{object name}\rangle 
FROM \langle\text{grantee}\rangle [[, \langle\text{grantee}\rangle]…] [\langle\text{CASCADE}\rangle] 

3.3.1.2 Conceptual Schema The DBA creates a table using the create statement:
\(\langle\text{table definition}\rangle\) ::= 
\text{CREATE} \left[\langle\text{GLOBAL|LOCAL} \text{T} \text{EMPORARY}\rangle\right] \langle\text{TABLE}\rangle 
\langle\text{maximum classification}\rangle 
\langle\text{default classification}\rangle 
\langle\text{table elements}\rangle 
[\langle\text{ON COMMIT} \langle\text{DELETE|PRE} \text{SERVE}\rangle \langle\text{ROWS}\rangle\rangle] 
\langle\text{primary key}\rangle 
\langle\text{maximum classification}\rangle ::= 
\langle\text{M} \text{AXIMUM L} \text{EVEL} \langle\text{classification level}\rangle\rangle 
\langle\text{classification level}\rangle ::= 
  \langle\text{literal}\rangle 
\langle\text{default classification}\rangle ::= 
\langle\text{DEFAULT L} \text{EVEL} \langle\text{classification level}\rangle\rangle 
\langle\text{table elements}\rangle ::= 
  \langle\text{table element}\rangle [[, \langle\text{table element}\rangle]…] 
\langle\text{table element}\rangle ::= 
\langle\text{column definition}\rangle 
\langle\text{table constraint definition}\rangle 
\langle\text{column definition}\rangle ::= 
\langle\text{column name}\rangle [[\langle\text{data type}\rangle\langle\text{domain name}\rangle] 
\langle\text{default classification}\rangle] 
\langle\text{derivation factor list}\rangle 
\langle\text{default clause}\rangle 
\langle\text{column constraint definition}\rangle [\langle\text{column collation}\rangle] 
\langle\text{data type}\rangle ::= 
\langle\text{character string type}\rangle 
\langle\text{character set\langle\text{character set specification}\rangle}\rangle 
\langle\text{national character string type}\rangle 
\langle\text{bit string type}\rangle 
\langle\text{numeric type}\rangle 
\langle\text{datetime type}\rangle 
\langle\text{interval type}\rangle 

\langle\text{derivation factor list}\rangle ::= 
\langle\text{derivation factor}\rangle [[, \langle\text{derivation factor}\rangle]…] 
\langle\text{derivation factor}\rangle ::= 
\langle\text{DERIVE L} \text{EVEL} \langle\text{classification level}\rangle\rangle 
\langle\text{FROM LEVEL} \langle\text{classification level list}\rangle\rangle 
\langle\text{classification level list}\rangle ::= 
\langle\text{classification level}\rangle [[, \langle\text{classification level}\rangle]…] 
\langle\text{default clause}\rangle ::= 
\langle\text{DEFAULT} \langle\text{default option}\rangle\rangle 
\langle\text{default option}\rangle ::= 
\langle\text{literal}\rangle 
\langle\text{date}\text{time value function}\rangle 
\langle\text{USER}\rangle 
\langle\text{SYSTEM USER}\rangle 
\langle\text{NULL}\rangle 
\langle\text{column constraint definition}\rangle ::= 
\langle\text{constraint name definition}\rangle 
\langle\text{column constraint}\rangle 
\langle\text{constraint name definition}\rangle ::= 
\langle\text{CONSTRAINT} \langle\text{constraint name}\rangle\rangle 
\langle\text{constraint name}\rangle ::= 
\langle\text{qualified name}\rangle 
\langle\text{column constraint}\rangle ::= 
\langle\text{NOT NULL}\rangle 
\langle\text{references specification}\rangle [[\langle\text{referential triggered action}\rangle] 
\langle\text{check constraint definition}\rangle 
\langle\text{references specification}\rangle ::= 
\langle\text{REFERENCE}\rangle \langle\text{referenced table and columns}\rangle 
\langle\text{MATCH} \langle\text{match type}\rangle\rangle 
\langle\text{referenced table and columns}\rangle ::= 
\langle\text{table name}\rangle [[\langle\text{reference column list}\rangle]] 
\langle\text{reference column list}\rangle ::= 
\langle\text{column name}\rangle [[, \langle\text{column name}\rangle]…] 
\langle\text{match type}\rangle ::= 
\langle\text{FULL}\rangle 
\langle\text{referential triggered action}\rangle ::= 
\langle\text{update rule}\rangle [[\langle\text{delete rule}\rangle]] 
\langle\text{delete rule}\rangle [[\langle\text{update rule}\rangle]] 
\langle\text{update rule}\rangle ::= 
\langle\text{ON UPDATE} \langle\text{referential action}\rangle\rangle 
\langle\text{referential action}\rangle ::= 
\langle\text{CASCADE}\rangle 
\langle\text{SET NULL}\rangle 
\langle\text{SET DEFAULT}\rangle 
\langle\text{delete rule}\rangle ::= 
\langle\text{ON DELETE} \langle\text{referential action}\rangle\rangle 
\langle\text{check constraint definition}\rangle ::= 
\langle\text{CHECK} \langle\text{search condition}\rangle\rangle 
\langle\text{column collation}\rangle ::= 
\langle\text{collate clause}\rangle 
\langle\text{collate clause}\rangle ::= 
\langle\text{COLLATE USING COLUMN} \langle\text{identifier}\rangle\rangle 
\langle\text{collate clause}\rangle ::= 
\langle\text{COLLATE} \langle\text{collation name}\rangle\rangle 
\langle\text{collation name}\rangle ::= 
\langle\text{qualified name}\rangle
For the definitions of (literal), (qualified name), (datetime value function), (character string type), (character set specification), (national character string type), (bit string type), (datetime type), and (interval type) we refer to [2].

The ANSI SQL2 (table definition) has been extended with (maximum classification), (default classification), (primary key) and (derivation factor list). The maximum classification defines the maximum classification level allowed for the table. The (default classification) defines a default classification level for a table and a column. It is required for a table but not for a column. It is not defined for a column, the default level given for the table is used. The (primary key) defines one or more columns to be used as a primary key. The (derivation factor list) defines a list of derivation factors where each derivation factor is denoted by (derivation factor). The (derivation factor) defines that the values for a column are to be taken from one or more rows that have been previously entered with the same primary key at the (classification level list) if possible. The (classification level list) defines an ordering on the rows to be used in deriving the value for a column. This construct is useful for polyninstantiation (note that by polyninstantiation we mean users at different security levels having different views of the same entity). For example, it allows a Top Secret user to derive values for a particular column from either Secret or Unclassified user rows, so that the TopSecret user does not have to reenter these values. Derivation is discussed further in the paper describing the complete design of LDV [11].

The DBA alters a table using the alter statement:

\[
\langle \text{alter table statement} \rangle \equiv \\
\text{ALTER TABLE } \langle \text{table name} \rangle \langle \text{alter action} \rangle
\]

\[
\langle \text{alter action} \rangle \equiv \\
\langle \text{add column definition} \rangle \\
\langle \text{drop column definition} \rangle \\
\langle \text{add table constraint definition} \rangle \\
\langle \text{drop table constraint definition} \rangle \\
\langle \text{replace default clause} \rangle
\]

\[
\langle \text{add column definition} \rangle \equiv \\
\text{ADD } \langle \text{column definition} \rangle
\]

\[
\langle \text{drop column definition} \rangle \equiv \\
\text{DROP } \langle \text{column name} \rangle \langle \text{CASCADE} \rangle
\]

\[
\langle \text{add table constraint definition} \rangle \equiv \\
\text{ADD } \langle \text{table constraint definition} \rangle
\]

\[
\langle \text{drop table constraint definition} \rangle \equiv \\
\text{DROP CONSTRAINT } \langle \text{constraint name} \rangle \langle \text{CASCADE} \rangle
\]

\[
\langle \text{replace default clause} \rangle \equiv \\
\text{REPLACE } \langle \text{column name} \rangle \langle \text{default clause} \rangle
\]

The DBA deletes a table using the drop statement:

\[
\langle \text{drop table statement} \rangle \equiv \\
\text{DROP TABLE } \langle \text{table name} \rangle \langle \text{CASCADE} \rangle
\]

The DBA defines discretionary access rights using the grant statement:

\[
\langle \text{grant statement} \rangle \equiv \\
\text{GRANT } \langle \text{privileges} \rangle \text{ ON } \langle \text{object name} \rangle \text{ TO } \langle \text{grantee} \rangle \langle \text{WITH GRANT OPTION} \rangle
\]

The DBA revokes discretionary access rights using the revoke statement:

\[
\langle \text{revoke statement} \rangle \equiv \\
\text{REVOKE } \langle \text{GRANT OPTION FOR} \rangle \langle \text{privileges} \rangle \text{ ON } \langle \text{object name} \rangle \text{ FROM } \langle \text{grantee} \rangle \langle \text{CASCADE} \rangle
\]

The DBA defines integrity assertions using the assertion statement:

\[
\langle \text{assertion definition} \rangle \equiv \\
\text{CREATE ASSERTION } \langle \text{constraint name} \rangle \langle \text{assertion check} \rangle \\
\langle \text{assertion check} \rangle \equiv \\
\text{CHECK } \langle \text{search condition} \rangle \\
\langle \text{assertion check time} \rangle \equiv \\
\text{DEFERRED } \langle \text{IMMEDIATE} \rangle
\]

The DBA deletes integrity assertions using the drop assertion statement:

\[
\langle \text{drop assertion statement} \rangle \equiv \\
\text{DROP ASSERTION } \langle \text{constraint name} \rangle
\]

Currently the design does not maintain general
semantic integrity. There are two steps involved in maintaining semantic integrity [8]:

1. Provide the DBA with a language for specifying semantic constraints.
2. Use the constraints to automatically check data that are presented for insertion into the database.

Our design provides the language, enforces one type of semantic integrity constraint that specifies legal domains for columns on insertion and update. There are two problems in enforcing general constraints. First, it is undecidable whether the constraints themselves are consistent. Second, the enforcement might require access to a large portion of the database. We recommend that general semantic integrity constraint enforcement be incorporated into LDV as research in this area progresses.

3.3.1.3 Internal Schema
The following extensions are necessary in order for the DBA to create indexes.

\[
\text{(index definition)} := \text{CREATE INDEX } \langle \text{index name} \rangle \langle \text{index column list} \rangle
\]

\[
\text{(index column list)} := \langle \text{column name} \rangle \| \langle \text{column name} \rangle \ldots
\]

The DBA deletes indexes using the drop index statement:

\[
\text{(drop index)} := \text{DROP INDEX } \langle \text{index name} \rangle
\]

3.3.2. DBSSO Language

3.3.2.1 Conceptual Schema
The DBSSO creates security assertions using the security assertion statement.

\[
\text{(security assertion)} := \text{ASSERT SECURITY } \langle \text{assertion name} \rangle \rightarrow \text{TO } \langle \text{security specification} \rangle \rightarrow \text{LEVEL } \langle \text{classification level} \rangle \rightarrow \text{ION } \langle \text{event list} \rangle
\]

\[
\text{(security specification)} := \text{SELECT } [\text{EACH OF } \langle \text{select list} \rangle \langle \text{table expression} \rangle
\]

The definitions of \( \langle \text{select list} \rangle \) and \( \langle \text{table expression} \rangle \) will be given in 3.3.3.1.

The ANSI SQL2 has been extended with \( \langle \text{security assertion} \rangle \). The \( \langle \text{security assertion} \rangle \) defines a classification constraint that classifies data based on content, context or aggregation. The \[ \text{EACH OF} \] construct is used to distinguish between a by context constraint, and a shorthand for more than one by content constraint. For example, the following constraint specifies that Name and Salary are TopSecret when taken together (a by context constraint):

\[
\text{ASSERT SECURITY S1 TO}
\]

\[
\text{SELECT Name, Salary FROM EMPLOYEE LEVEL TopSecret}
\]

The following constraint specifies that name and Salary and TopSecret when taken individually:

\[
\text{ASSERT SECURITY S1 TO}
\]

\[
\text{SELECT Name, Salary FROM EACH OF EMPLOYEE LEVEL TopSecret}
\]

This constraint is a shorthand for the following two constraints:

\[
\text{ASSERT SECURITY S1a TO}
\]

\[
\text{SELECT Name FROM EMPLOYEE LEVEL TopSecret}
\]

\[
\text{ASSERT SECURITY S1b TO}
\]

\[
\text{SELECT Salary FROM EMPLOYEE LEVEL TopSecret}
\]

The DBSSO deletes security assertions using the drop security assertion statement:

\[
\text{(drop security assertion)} := \text{DROP ASSERTION } \langle \text{assertion name} \rangle
\]

3.3.3. Ordinary User Language

Ordinary users use the DML against the tables that have been defined in the external schema. The ANSI standard DML is presented below. The DML has been extended with a time-oriented construct that refers to points in time, and a level-oriented construct that refers to the classification level of data.

3.3.3.1 Data Manipulation Language
An ordinary user retrieves data using query specification statement:

\[
\text{(query specification)} := \text{SELECT } [\text{ALL | DISTINCT}]
\]

\[
\text{(select list) \langle table expression \rangle}
\]

\[
\text{(select list)} := \text{\langle select sublist \rangle | \{ \langle select sublist \rangle \ldots \}
\]

\[
\text{(select sublist)} := \text{\langle derived column \rangle \langle qualifier \rangle \ast}
\]

\[
\text{(derived column)} := \text{\langle value expression \rangle}
\]

\[
\langle \langle \text{column name} \rangle \rangle
\]

\[
\text{(value expression)} := \text{\langle numeric value expression \rangle}
\]

\[
\text{\langle string value expression \rangle}
\]

\[
\text{\langle datetime value expression \rangle}
\]

\[
\text{\langle interval value expression \rangle}
\]

\[
\text{\langle numeric value expression \rangle := \langle term \rangle \ast}
\]

\[
\text{\langle numeric value expression \rangle + \langle term \rangle}
\]

\[
\text{\langle numeric value expression \rangle – \langle term \rangle}
\]
The ANSI SQL2 (primary) has been extended with (data characteristic function specification). The (data characteristic function specification) allows the user to refer to the level and timestamp of columns. Using these extensions, the user can select the classification level and timestamp of data, and can select data based upon its classification level and timestamp. For example, the following query selects the level and timestamp of data: SELECT Name, LEVEL(Name) TIMESTAMP(Name) FROM EMPLOYEE
and the following query selects data based upon the level and timestamp:
The ordinary user inserts data using the insert statement:
INSERT INTO (table name) [(insert column list)]
=query expression) ;
(insert column list):= (column name) [{, (column name)} ...]
The ordinary user deletes data using the delete statement.
DELETE FROM (table name) WHERE CURRENT OF (cursor name)
(delete statement: searched):= DELETE FROM (table name) [WHERE (search condition)]
The ordinary user updates data using the update statement:
UPDATE (table name) SET (set clause: positioned) [{, (set clause: positioned)} ...] WHERE CURRENT OF (cursor name)
(set clause: positioned):= (object column: positioned) = [{ (value expression) I (null specification) }]
(object column: positioned):= (column name)
(update statement: searched):= UPDATE (table name) SET (set clause: searched) [{, (set clause: searched)} ...] WHERE (search condition)
(set clause: searched):= (object column: searched) = [{ (value expression) I (null specification) }]
(object column: searched):= (column name)
For the definition of (cursor name) we refer to [2].
In summary, the extensions to the ANSI SQL2 standard for each user group are as follows:
The ordinary user sees small changes in the ANSI SQL2 standard. He sees changes in the SELECT statement to allow for use of the data characteristics of LEVEL and TIMESTAMP in the select list WHERE clause.
The DBA sees more changes in the ANSI SQL2 standard. He sees changes in the definition of a table. He must define maximum classification,
default classification, and primary key for the table. He can define default classifications and derivation factors for individual columns. In addition, he can create and delete indexes.

The DBSSO is a new role not present in non-secure database environments. The DBSSO is responsible for the mandatory security of the database as it relates to disclosure of information, while the DBA is responsible for the structure and organization of the database. The DBSSO can create and delete security constraints using SQL.

4. Conclusion

We have described the essential features of a multilevel secure relational database system and discussed extensions to the relational language SQL that must be supported. These extensions are necessary to handle the security assertions which assign security levels to the data. Future work in this area will include specifying SQL extensions to support many different types of mandatory as well as discretionary assertions.

We have completed the design of LDV and hope to implement it in the near future. We expect to use a commercial relational database system for the near-term implementation. SQL preprocessor and parser have to be modified to support the extensions that we have suggested.

Acknowledgement

This work was supported by Rome Air Development Center under RADC contract No: FQE30602-86-C-0003. RADC's support is gratefully acknowledged. The other members of the Honeywell team included Patricia Dwyer, Thomas Haigh and Emmanuel Onuegbe, whom we also thank.

References